Thesis word count: 58720

COPYRIGHT STATEMENT

An Insider Misuse Threat Detection and Prediction Language

by

Georgios Vasilios Magklaras

This dissertation is submitted to the University of Plymouth

In fulfilment of the award of

DOCTOR OF PHILOSOPHY

School of Computing and Mathematics

Faculty of Science and Technology

July 2011

Abstract

An Insider Misuse Threat Detection and Prediction Language

Georgios Vasilios Magklaras BSc (Hons) MPhil

Numerous studies indicate that amongst the various types of security threats, the problem of insider misuse of IT systems can have serious consequences for the health of computing infrastructures. Although incidents of external origin are also dangerous, the insider IT misuse problem is difficult to address for a number of reasons. A fundamental reason that makes the problem mitigation difficult relates to the level of trust legitimate users possess inside the organization. The trust factor makes it difficult to detect threats originating from the actions and credentials of individual users. An equally important difficulty in the process of mitigating insider IT threats is based on the variability of the problem. The nature of Insider IT misuse varies amongst organizations. Hence, the problem of expressing what constitutes a threat, as well as the process of detecting and predicting it are non trivial tasks that add up to the multi-factorial nature of insider IT misuse.

This thesis is concerned with the process of systematizing the specification of insider threats, focusing on their system-level detection and prediction. The design of suitable user audit mechanisms and semantics form a Domain Specific Language to detect and predict insider misuse incidents. As a result, the thesis proposes in detail ways to construct standardized descriptions (signatures) of insider threat incidents, as means of aiding researchers and IT system experts mitigate the problem of insider IT misuse.

The produced audit engine (LUARM – Logging User Actions in Relational Mode) and the Insider Threat Prediction and Specification Language (ITPSL) are two utilities that can be added to the IT insider misuse mitigation arsenal. LUARM is a novel audit engine designed specifically to address the needs of monitoring insider actions. These needs cannot be met by traditional open source audit utilities. ITPSL is an XML based markup that can standardize the description of incidents and threats and thus make use of the LUARM audit data. Its novelty lies on the fact that it can be used to detect as well as predict instances of threats, a task that has not been achieved to this date by a domain specific language to address threats.

The research project evaluated the produced language using a cyber-misuse experiment approach derived from real world misuse incident data. The results of the experiment showed that the ITPSL and its associated audit engine LUARM provide a good foundation for insider threat specification and prediction. Some language deficiencies relate to the fact that the insider threat specification process requires a good knowledge of the software applications used in a computer system. As the language is easily expandable, future developments to improve the language towards this direction are suggested.

AUTHOR'S DECLARATION

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

Relevant conferences and security events were attended during the course of the research. In addition, system prototype source code and several papers were prepared for publication in refereed international journals and conferences, details of which are listed in the Appendices.

Signed ....

Date ....01/06/2011....

List of Contents

LIST OF FIGURES

ACKNOWLEDGEMENTS

This journey started when a young (and rather arrogant) graduate decided to enter the world of Information Security through an academic door. He went through a series of learning experiences of pleasant and (at times) turbulent nature. He completed the journey because he eventually controlled his arrogance and received help from many people.

First and foremost, I would like to thank my supervisors Prof. Steven Furnell and Dr. Maria Papadaki, not only for their valuable guidance role but also for being patient with me and for helping me to engage into a PhD whilst having a demanding full time job, learning the ropes of domain specific language engineering and insider threat specification.

The help of Professor Kjetil Taskén and Ragni Indahl of the Biotechnology Center of Oslo, University of Oslo was vital for the completion of this thesis, as they have been the most reasonable employers one could ask for. The same goes for my team members, IT engineers Melaku Tadesse, Harald Dahle and Jean Lorentzen who helped me with some of my experiments and covered for me on many occasions at work.

I wish to extend my thanks to Professor Christian Probst (Technical University of Denmark), Professor Dieter Gollmann (Hamburg University of Technology, Germany) and Becky Bace for giving me helpful feedback during the Schloss Dagstuhl Insider Threat seminars. I am also grateful to Dr. Phil Brooke for providing valuable guidance in the field of XML parsers.

To the beta testers of LUARM, who proved that the open source community is one of the best places to develop software, goes a big thank you.

My heartfelt thanks go also to my wife Lucia and the rest of my family who gracefully faced the stressed side of me in my effort to complete this project.

Oslo, May 2011

Georgios V. Magklaras

In Memoriam of my colleague Dr. Harjit Singh and my teacher Georgios Alexandropoulos.

They both inspired me and left very early.

“ΟΣΟΙ ΔΕ ΒΟΥΛΗΣΟΝΤΑΙ ΤΩΝ ΤΕ ΓΕΝΟΜΕΝΩΝ ΤΟ ΣΑΦΕΣ ΣΚΟΠΕΙΝ ΚΑΙ ΤΩΝ ΜΕΛΛΟΝΤΩΝ, ΠΟΤΕ ΑΥΘΙΣ ΚΑΤΑ ΤΟ ΑΝΘΡΩΠΕΙΟΝ ΤΟΙΟΥΤΩΝ.¨

“To those that wish to have a precise perception of the events, those that have already occurred and those (events) that due to human nature are going to occur”

Thucydides history, A22

Chapter 1 Introduction

Information Technology (IT) security threats concern every component of the modern computing infrastructure world. Pfleeger [1] defines the term threat in an IT infrastructure context as “a set of circumstances that has the potential to cause loss or harm”. These circumstances might involve human-initiated actions (intentional IT intrusions), flaws in the design of the computer system and environment factors (natural disasters). Electronic commerce and banking, telecommunications, air traffic control [2] and public transportation, energy and water distribution and electronic voting systems [3] are some of the examples that emphasize our increasing dependency on computing infrastructures and thus highlight the potentially serious impact of IT security threats for the normal functioning of our society.

Various sources indicate the growing trend of IT security threats, including books [4] [5] and surveys [6] [7] that reveal the technical, as well the social and economic implications of IT threats. The recorded trends indicate that during the last decade, there was a sharp rise in the number of security breaches that originated from external (i.e. unauthorized users) sources. Proprietary information theft from large enterprises with serious economic consequences, embarrassing web site defacements and denial of service (DoS) attacks that cripple the ability of IT to function are some of the examples that IT security specialists have to deal with.

A second and more recently highlighted trend of IT threat occurrence concerns authorized users of IT systems, commonly known as insiders. These users abuse their privileged access rights by committing a series of unintentional or deliberate actions, damaging individuals or organizations in many different ways. The dissemination and storage of offensive material through e-mail and the stealing of proprietary information for rival companies are probably the most traditional cases of insider IT misuse that one observes at the time of writing.

Despite the well-documented seriousness and emergence of insider threat [8], the information security world lacks approaches that mitigate the problem to a satisfactory extent. Insider IT misuse is a challenging multi-factorial problem. It encompasses aspects of technical issues (system monitoring, detection and prediction models), human issues (human resource policies, valuing and trusting individuals), organizational issues (defining what constitutes a threat, what is the value of your trusted information) and legal/ethical issues (how intrusive can you be/need to be, in order to have an adequate picture that detects or predict insider threat). As a result, it is not a surprise that a panacea to address all aspects of the problem does not exist.

This thesis focuses on certain technical aspects of the insider misuse problem. In particular, it addresses the vital issue of Insider Threat Specification and Prediction. One of the most fundamental problems of the insider threat domain is to define what constitutes a threat at system-level, beyond the scope of a security policy. This was one of the conclusions derived from the author’s prior MPhil research entitled 'An Architecture for Insider Misuse Threat Prediction in IT Systems' [9]. In order to illustrate the importance of insider threat specification, let us assume that an organizational security policy states the following in plain language terms:

“*It is not allowed for employees to use the company computer infrastructure to access and disseminate pornographic material.*”

The enforcement of this policy assumes a mechanism able to express in a standard way how the act of accessing and disseminating pornographic material manifests itself in system terms. In that way, it is possible to intercept file access and network connections events, as well as applications that access network connections and thus make a concrete link amongst all these pieces of information and a particular threat scenario.

An ability to use a standard vocabulary to define a threat in terms of its observable consequences at the Operating System (OS) level forms the core of the proposed Insider Threat Specification and Prediction Language (ITPSL). At the time of writing, generic insider misuse specification languages do not exist, highlighting the novelty of this research project for detecting insider threats. The proposed language can also act as the foundation for creating Insider Threat signature repositories and facilitate the reuse of threat descriptions (with minor adaptations) on different IT infrastructures.

A second novel aspect of this research project concerns the ability to predict insider threats. Threat prediction is closely related to threat modeling. Later chapters introduce the concept of insider threat modeling. For the purposes of introduction, it is sufficient to state that the proposed language semantics need to encompass decision theoretic information. The process of embedding decision theoretic information inside intrusion specification semantics is an important issue of the research field that has not been addressed.

Finally, LUARM, the bespoke audit engine designed to log relevant user activities for the purposes of misuse detection is unique in its kind, due to the way it structures the data, the format of the audit records and its usefulness to the computer forensics fields. A researcher using the proposed language semantics and the data acquired by LUARM can replay certain hypothetical or real scenarios for the purposes of refining misuse detection tools and threat signatures.

1.1 Aims and Objectives

This thesis is concerned with the process of systematizing insider threat specification and prediction. The overall aim is to advance the field of insider IT misuse research by producing a misuse specification language that can aid researchers and IT practitioners in the process of detecting and predicting insider threats. The following objectives of equal priority apply, as a guide to break down the task of crafting the proposed language:

- **Objective 1**: Investigate the problem of insider misuse, encompassing the general nature and scale of the threat and examination of the resulting impacts by means of indicative case examples.
- **Objective 2**: Define the problem of insider threat specification and prediction, as part of the insider IT misuse problem domain. This definition should emphasize the underlying main components of both the insider misuse detection and prediction processes and examine how they relate to existing work of the research field.
- **Objective 3**: Define the functional requirements of the proposed language. In plain terms, this means that it should be clear what the language should do and how it should achieve it.
- **Objective 4**: Establish the means of acquiring data about user credentials and on-line actions to facilitate threat detection and prediction. This can be achieved by reviewing existing operating system audit mechanisms in accordance to the functional requirements of Objective number 2.
- **Objective 5**: Produce the semantics and the syntax for the proposed language.
- **Objective 6**: Make a prototype system and evaluate the proposed language against a range of hypothetical and real insider IT misuse scenarios.

1.2 Thesis structure

After establishing the objectives of the research project, the thesis begins by tackling the problems of Objective 1. Chapter 2 provides a summary of the Insider IT misuse problem as well as an overview of notable insider misuse cases and information security surveys, in an attempt to show the manifestation of the problem in IT infrastructures. The chapter concludes with a discussion of relevant specification language paradigms and frameworks that influenced the development of ITPSL and thus complete a high-level overview of the problem domain.

Chapter 3 starts by providing an essential definition for the term 'insider threat specification', the core concept of this research project. The problem of insider threat specification relates to the field of misuse detection languages. As a result, a detailed discussion of earlier misuse detection language paradigms is presented, accompanied by proposing the desired semantic foundations for the proposed language (ITPSL). The combined contents of chapters 2 and 3 will address Objective 1 of this project, completing the definition of the insider threat specification and prediction problem domain.

The fourth chapter is concerned with taxonomic and threat modelling research and development efforts designed to address insider threats, with emphasis on abstracting the domain of insider misuse. The domain abstraction is a necessary step in the process of shaping the threat metrics the language can express and pave the way for defining the ITPSL functional requirements. After concluding the domain abstraction considerations, the chapter concludes by explaining the problems that the ITPSL is trying to solve and presents a detailed list of functional requirements, completing Objective 2 of the thesis.

Chapter 5 takes the first practical implementation step and examines the vital issue of how to store user activity information in audit logs. Unsuitable information in the audit logs can seriously hinder the process of insider threat specification and prediction. Thus, existing audit log mechanisms are examined in terms of their deficiencies for insider threat detection and prediction. The mentioned deficiencies justify the need for constructing LUARM, an audit log engine specifically designed for insider misuse detection and prediction. The construction of LUARM marks the completion of the third objective of the research project.

Chapter 6 contains the core proposal of the ITPSL semantics, addressing the fourth main objective of the thesis. It starts with a justification of the programming and semantic encapsulation paradigms used in the construction of ITPSL. In particular, an XML based Domain Specific Language (DSL) paradigm is chosen as the foundation of the proposed language. The rest of the chapter discusses in great detail the ITPSL semantics.

Having produced the proposed language semantics, the final objective of the research is addressed by putting the semantics into action. A prototype ITSPL compiler is presented in Chapter 7. The system provides the ability to use collected data by LUARM. The data are produced by carefully crafted experiments that simulate a range of insider IT misuse threats. These data are then played back against ITPSL statements, in order to assess the ability of the semantics to express a range of scenarios successfully. The ITPSL expression success is judged in terms of detecting and predicting certain threats under the conditions of the experiment.

The eighth and final chapter of the thesis provides a critique of the work, addressing weaknesses and future research issues related to insider threat specification and prediction.

The thesis appendices provide a plethora of detailed references to relevant technology standards, experiments, as well as copies of publications associated with this research project.

Chapter 2 The insider IT misuse problem

2.1 Definition of the problem domain

The problem of insider IT misuse (the term ‘misuse detection’ or ‘misuse’ is also used in the literature) is a serious threat for the health of IT infrastructures. The introductory chapter defined the term 'threat' in an IT infrastructure context as “a set of circumstances that has the potential to cause loss or harm” [1]. As a result, in legitimate user context, these circumstances might involve intentional IT misuse activities such as targeted information theft, introducing or accessing inappropriate material, and accidental misuse (e.g. unintentional information leak). In addition, there is also potential for flaws in the design and implementation of the computer system, which could render it susceptible to insider misuse.

Numerous studies have tried to define the term “insider” in the context of Information Security. This is because there are many possible sub-contexts that are applicable to shedding light on different aspects of what an insider is and what she can do. For instance, an aspect of insiders relates to what they are allowed and not allowed to do in an organizational context. This is often dictated by the organization's IT usage policy, “a set of laws, rules, practices, norms and fashions that regulate how an organisation manages, protects, and distributes the sensitive information and that regulates how an organisation protects system services” [10]. This is also commonly referred to as the IT security policy. Insiders that do not follow the rules of the IT policy are formally considered as misusers.

Other definitions focus more on the attributes of an insider, from an organizational trust point of view [11]:”An insider is a person that has been legitimately empowered with the right to access, represent, or decide about one or more assets of the organization's structure”. This definition has a wide perspective and emphasizes a key aspect of an insider: that of trust. Trust is a property that goes beyond an IT system oriented view (system credentials, actions, indications). Whilst people who constitute direct threats might not have access to IT access credentials, they still can decide on policies, equipment procurement and other issues that can affect the well being of an IT infrastructure. A good example is an IT director that spends millions on a state of the art security system but does not bother to emphasize or make policies that dictate the flow of information inside the organization (employee that bypasses the system with a simple USB key, intentionally or accidentally).

However, trust has an impact on IT level credentials. A narrower but IT system specific definition can also be useful, in order to focus on insider actions that can be detected by system methods. Hence, an insider is a person that has been legitimately given the capability of accessing one or many components of an IT infrastructure (hardware, software and data) enjoying effortless login by interacting with one or more authentication mechanisms. The word 'legitimately' differentiates the user from an external cracker that masquerades as the user by means of bypassing the authentication mechanisms. The implication of 'effortless' is that an insider does not need to consume time and effort to gain access to a system resource. This also means that they enjoy trust, a vital property of all insiders.

2.2 The manifestation of the insider IT misuse problem

Providing a way to detect threats and sense vulnerabilities is vital for the process of insider threat mitigation. One way to capture the essence of insider threats is to look at the way they occur in the real world. There are two sources of information to help us derive conclusions. One consists of insider case reports, as they are reported by the press. The other source of information is an established information security survey. Both sources have pros and cons and we will discuss these in this section.

One of the most discussed insider cases is that of the former FBI veteran Robert Hanssen. The 2001 CSI/FBI survey [12] cited his case in detail. Hanssen abused his trusted access to the FBI Automated Case Support System that contained classified information about ongoing investigations and handed critical information to Russian agencies. In return, he was receiving large sums of money, inflicting a great deal of damage upon the prestigious image of the Federal Bureau of Investigation and the national security of his country. Nobody could imagine that a church-going and patriotic family man was betraying his country for money.

Hanssen developed a more than average level of IT knowledge, as he utilised an unusual way of hiding the information he wanted to trade with Russian agents. He followed a process of specially formatting 40-track mode diskettes, in order to hide the sensitive information in (what appeared to be) a blank area of the disk. This measure made it difficult to discover the hidden insider view information without the usage of an advanced data forensic tool. The combination of his colleagues' trust and his own data hiding techniques allowed him to operate for certain number of years inside various FBI facilities.

Abdelkader Smires [13], a chief software engineer who worked with Internet Trading Technologies is a typical example of what can be achieved by a disgruntled insider. Smires claimed he was underpaid by his employer. As a result, he requested a pay rise coupled with a range of additional benefits. When his requests were turned down, he decided to take revenge by using the IT infrastructure of his previous employer (Queens College) to launch a Denial of Service (DoS) attack. His actions crippled the operational capacity of Internet Trading Technologies over a three day period resulting in substantial revenue loss.

There are two important points to consider with regards to the Smires case. Firstly, he had legitimate access to another organisation (Queens College) due to an account that should had been erased a long time ago. This shows how a bad insider practice (keeping non essential active accounts is bad system administration practice) allowed him to conceal (at a first stage) his attack on Internet Trading Technologies. The second and most important point is the level of systems knowledge he possessed about Internet Trading Technologies´ IT infrastructure. Smires´ excellent knowledge of the IT components catalysed his ability to exploit vulnerabilities and mount the DoS attack.

The financial world with its business critical cyberinfrastructures is not immune to insider misuse cases. Garfinkel and Spafford [14] mention the „Leeson-Iguchi‟ case. Nick Leeson („Barings‟ Bank – Singapore) and Toshihibe Iguchi („Daiwa Bank‟ - New York) were investment traders working together for two major financial organisations. They made risky investments and lost large amounts of investment capital. However, instead of admitting their losses, they illegitimately modified computer records to cover their mistakes and continue to be able to request vast amounts of money to invest. As a result, Barings Bank was forced to insolvency and „Daiwa‟ lost its entire United States customer base. More than 1 billion dollars of investment capital vanished as a result of their actions.

Barings Bank had an internal data audit mechanism that focused on discovering potential external breaches, without focusing on insider actions. Clearly, they have underestimated the insider threat factor. They could never think that two accountants that had direct access to database records of investment funds would commit fraud in this way. This electronic record forgery would probably go unnoticed if Leeson and Iguchi managed to stop their losses. They did not and consequently the large sums of unaccounted investment capital forced an internal investigation that revealed their actions.

All of the previously discussed cases were intentional. Legitimate users misused their rights on purpose to achieve the goal. However, non intentional insider misuse cases are also part of the manifestation agenda. The Norwich Union versus Western Provident Association case [15] is a classic case example of accidental insider misuse. A Norwich Union employee circulated an e-mail that contained what could be considered as a sarcastic (or defamatory) rumour about Western Provident going into financial difficulties. The e-mail leaked outside the company (another internal user thought it was a great joke) and eventually came to the attention of the rival company. Consequently, Western Provident took legal action against Norwich Union and the case was settled with the latter paying approximately £450,000 pounds in compensation plus the legal expenses for Western Provident. What was initially considered as an innocent joke proved to be the reason for commencing a rather expensive legal case.

The most widely known insider misuse cases are usually about intellectual property theft. The arrest of Lan Lee and Yuefei Ge by FBI agents [16] is a classic case. The arrested men were engineers of NetLogic Microsystems (NLM) until July 2003. During the time of their employment, they were downloading trade sensitive documents from the NLM headquarters into their home computers. These documents contained detailed descriptions of the NLM microprocessor product line and funded a startup made by the two engineers. Eventually, their ties to the Chinese government and military were discovered by investigators.

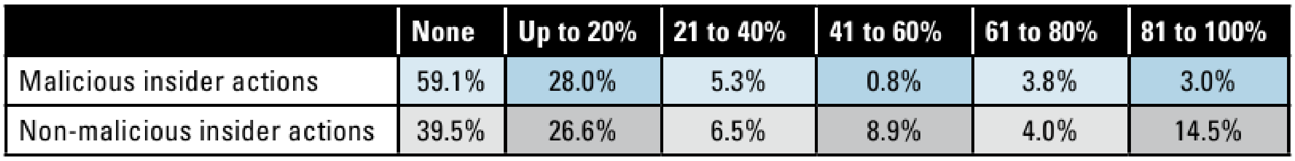
Establishing borders between internal and external cases can be difficult. It is not suggested that one should adopt a dualistic view and classify an incident as either internal or external one. An information security incident has often many factors. However, the key question in many cases is what creates the vulnerabilities. In this case, insiders accidentally created vulnerabilities leaving the gates open, despite the confirmed external attack origin. Consequently, the gravity of the actions of insiders is greater. This makes one wonder how organizations should assess or model threats in the future. For every virus attack, should an IT manager that left systems without proper anti-virus protection or system patches be blamed instead of the virus distributor?

There are numerous other cases reported in the press that follow more or less the same line in terms of the motives, the abuse of trust and cyber-privileges or the unintentional actions that ended up having grave consequences. However, a study of the insider threat manifestation should also include some quantitative measures that ideally show the frequency of occurrence in the real world. Well established information security surveys attempt to provide that level of information and the next paragraphs are going to discuss them.

Recent information security surveys document the existence and the seriousness of the problem. The Computer Security Institute's “Computer Crime A Security Survey” is a well cited survey that has been running for fifteen consecutive years. The 2010 survey [6] reviewed the opinions of 351 IT security practitioners. Amongst the key findings of this survey was that the insider threat has two important vectors:

- **Intentional (malicious) insiders**: Concerning IT misuse incidents that were the results of deliberate actions (for example the Robert Hansen case discussed in [12].
- **Non malicious (accidental) insiders**: They concern cases where the insider accidentally caused an IT misuse incident (a good example of this are users that entrust their account password to third party individuals for convenience).

Figure 2.1 below indicates the percentage of monetary losses reported by the respondents attributed to malicious and non malicious insiders that represents one of the key findings of the survey. 59.1 percent of the respondents believe that non of their losses were due to malicious insiders. However, only 39.5 percent could claim that none of their losses were due to non-malicious insider actions, hinting that the consensus views the accidental insider threat vector as a more serious one.



	None	Up to 20%	21 to 40%	41 to 60%	61 to 80%	81 to 100%
Malicious insider actions	59.1%	28.0%	5.3%	0.8%	3.8%	3.0%
Non-malicious insider actions	39.5%	26.6%	6.5%	8.9%	4.0%	14.5%

Figure 2.1: Percentage of monetary losses attributed to insiders (source [6])

In terms of frequency of occurrence, the CSI survey [6] reports that 25 percent of their respondents claimed that their legitimate users deliberately abused Internet access or email (pornographic access, pirated software, etc). In addition, 13% percent claimed that they detected cases of unauthorized privilege escalation by insiders (insiders abusing/bypassing their access rights to obtain information that should have been restricted to them).

The manifestation of insider IT misuse is also verified by another well established survey in the field of information security. The InfoSecurity Europe and PwC '2010 Information Security Breaches' (ISBS) survey [7] collected results from 539 respondents and shows a substantial rise of serious internal incidents when compared to the figures of previous years, as shown in Figure 2.2 below.

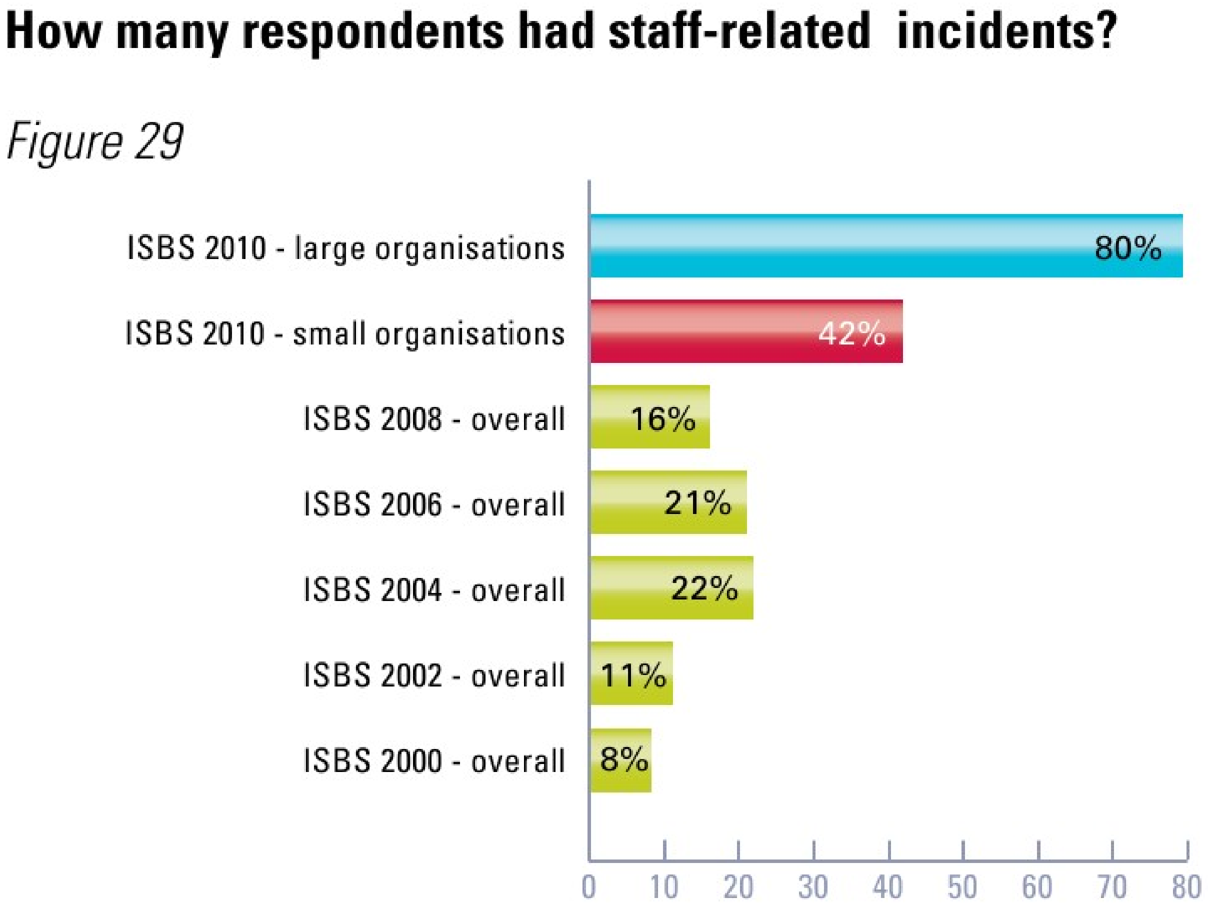


Figure 2.2: Insider incidents for the period 2000-2010 (source [7])

The ISBS 2010 survey looks into internal misuse cases in more detail than the CSI one [6]. In particular, the survey section with title 'Other incidents caused by staff' sketches the typical profile of misuse cases. The survey respondents indicated that an insider is more likely to use the world wide web to access inappropriate sites (or sites that have nothing to do with work duties, thus wasting resources). The second most frequent type of misuse is that of email (use email for non work purposes, to spread inappropriate or offensive content), followed by unauthorized access to systems and data (by means of masquerading or bypassing authentication procedures). Finally, data theft (breach of confidentiality of intellectual property or other valuable data) and breach of data protection rules and regulations are the two least frequent types of misuse reported. This picture is summarized in Figure 2.3.

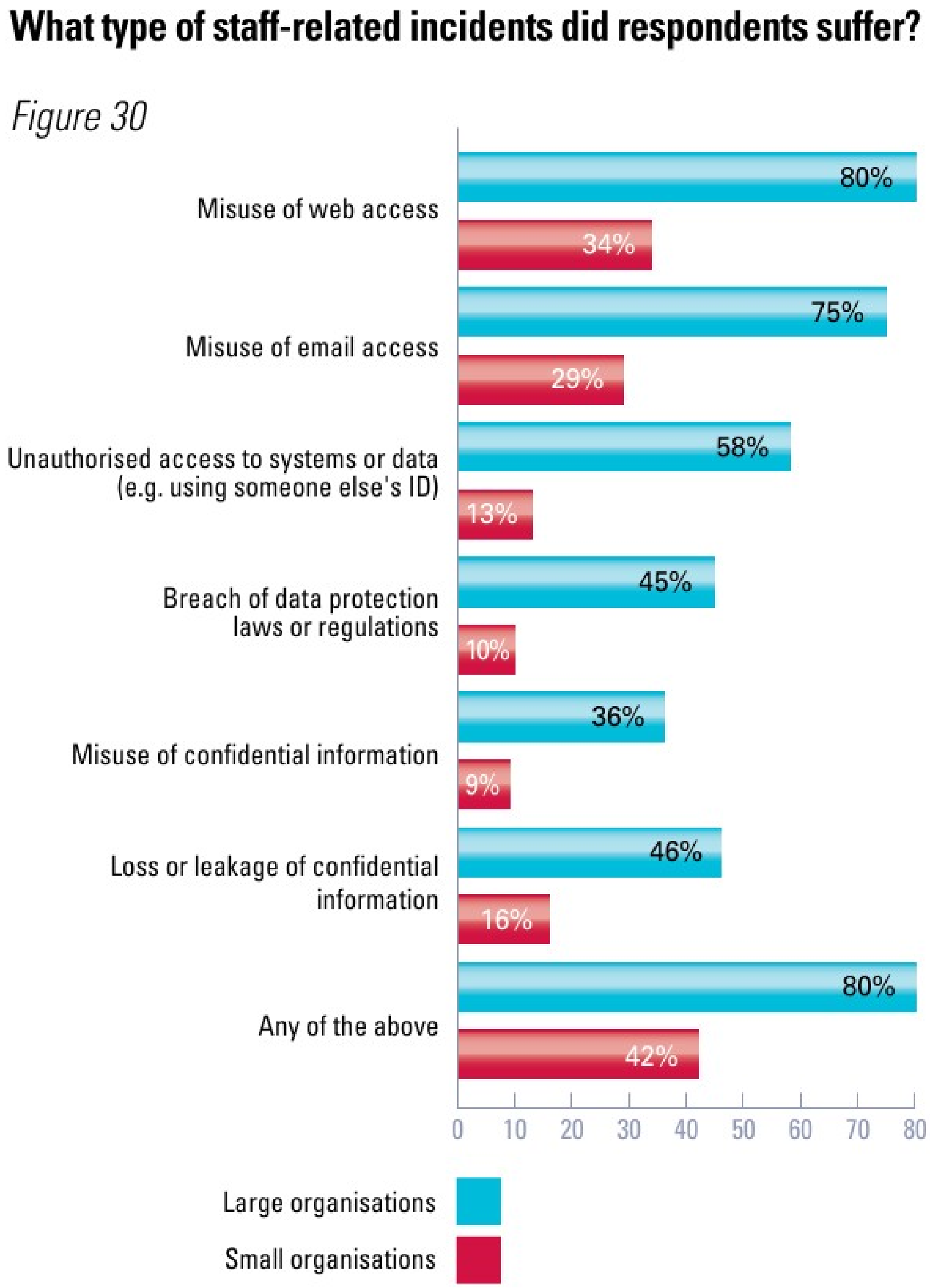


Figure 2.3: Type of insider misuse (source [7])

Similar conclusions about the nature (type and frequency) of insider misuse incidents are also derived by Magklaras and Furnell [17] , in a survey dedicated to insider IT misuse. Due to the similarity of the results and the fact that the survey had a rather small number of respondents (50), the detail is not discussed here, but the reader is prompted to reference the data for further information about the findings.

The ISBS 2010 survey [7] also provides some notable cases that verify the CSI 2010 [6] findings in terms of the seriousness of accidental insider threats. Some of these cases are quoted below, in order to demonstrate the grave consequences of non deliberate insider actions:

“Staff at a London educational institution replied to a phishing email. This resulted in spammers sending over 100,000 emails from the compromised accounts, and to the organization’s mail servers being blacklisted around the world.”

“A charity infringed data protection laws when it disposed of an old computer without wiping the hard drive. The staff member concerned was blasé, saying he had deleted the files and trusted the person to whom he had sold the computer.”

The conclusion from digesting the data from various surveys is that one cannot derive safe conclusions about the quantification of the insider IT misuse problem (frequency of occurrence, financial impact). The CSI 2010 survey [6] claims a statistical confidence of 95 percent with an associated +/- 5.25 percent margin of error for the quoted figures. The respective figures for the ISBS 2010 survey [7] are also high: a 95 percent confidence rate with +/- 6% error margin. These figures might be true for the sample population of the surveys, however they do not account fully for the following facts:

- **The sample population of the surveys is different between the two surveys**: A different number of respondents from different geographical populations (and hence different IT regulations) makes it difficult to compare the quantified figures and make conclusions about the true scale of the problem.
- **The sample population varies significantly on every survey edition**: In their latest edition at the time of writing, both of the quoted large scale surveys [6] [7] record a notable drop in the number of respondents coming from the financial and private sector. This could affect substantially the scale of the insider problem (number of incidents).

For these reasons, surveys are good in qualifying the problem and enlighten on the various different types of insider misuse manifestation. That's despite the important differences recorded amongst major surveys about the frequency of occurrence and perceived importance of insider incidents, which could be attributed due to differences in their respondent sampling procedures. In comparison, reported cases in the press provide some aspect of problem qualification, but they often omit important aspects of the case, things that could be of use to insider misuse researchers. Nevertheless, the data do converge into various conclusions:

- The insider problem is real, it affects all types of businesses in a serious manner and is here to stay.
- The insider threat has two important types: Deliberate and accidental misuse.
- A typical insider according to the surveys respondents is likely to abuse the web and email resources and manipulate authentication or other internal systems to commit fraud or information theft.
- The nature of the insider misuse problem is variable. Different organizations have different criteria of what is considered misuse or not that are applicable to different environments.

2.3 Intrusion specification languages and frameworks

While the previous chapter sections provided an overview of the problem domain, the focus must be shifted to intrusion specification. One of the conclusions of the previous section was that insider IT misuse is a variable problem. Neither the information security surveys nor the stories in the press can provide a clear picture of the mechanism with which the problem manifests itself in IT infrastructures, This clear picture should ideally display a mechanism that shows how a threat is realized into a misuse act. The field of intrusion specification provides fundamental tools to obtain such a picture.

ITPSL is all about insider threat specification. Threat specifications follow the principles of intrusion specification, a concept which is not new in the information security world. Techniques to describe threats exist for an entire range of information security products, from anti-virus software to several intrusion detection/prevention systems (IDS/IPS) [17], where specified rules are used to describe a particular range of threats. However, this section focuses on generic threat specification. Most products might focus on specific types of threats (anti-virus products relate to malware detection, IDS products might focus on network threats, etc).

When it comes to generic intrusion specification languages, we have two notable examples. The Common Intrusion Specification Language (CISL) [17] and Panoptis [18]. The next paragraphs are going to describe these two languages and discuss their significance for ITPSL.

CISL[17] consists of a semantic framework to unambiguously describe intrusive activities together with proposed data structures that store the event information and can form standardized messages exchanged by various Intrusion Detection System (IDS) components. The semantic representation of intrusive activities is achieved by the formation of an S-Expression. This is a recursive grouping of tags and data, delimited by parentheses. The tags provide semantic clues to the interpretation of the S-Expression and the data might represent system entities or attributes. For this reason, the tags are also called Semantic Identifiers (SIDs).

The best of way of illustrating how CISL works is by considering an example. The statement (Hostname 'frigg.uio.no') is a simple S-Expression. It groups two terms, without semantically binding them. One can guess that it refers to a computer system with the FQDN name 'frigg.uio.no', but the true meaning of the statement is still vague. In fact, the full semantic meaning of S-Expressions becomes apparent when one forms more complex S-Expressions, by means of combining several SIDs into a sentence.

Figure 2.4 illustrates a suitably crafted CISL intrusion specification which could be translated in the following plain English translation:
"*On the 24th of February 2005, three actions took place in sequence in the host 'frigg.uio.no'. First, someone logged into the account named 'tom' (real name 'Tom Attacker') from a host with FQDN 'outside.firewall.com'. Then, about a half-minute later, this same person deleted the file '/etc/passwd' of the host. Finally, about four-and-a-half minutes later, a user attempted but failed to log in to the account 'ksimpson' at 'frigg.uio.no'. The attempted login was initiated by a user at 'hostb.uib.no'.*"

The particular CISL sentence describes a malicious attack that erases an important system file of a UNIX system and consists of three multi-SID S-Expressions. In general, a sentence can be formed by one or more S-Expressions nested at different levels.

Verb SID's are joined together in a sentence by conjunction SIDs. In the example of Figure 2.4, 'And' is the conjunction SID that holds together the three SIDs that form the sentence. In addition, a CISL sentence might employ role, adverb, attribute, referent and atom SID types. Role SIDs indicate what part an entity plays in a sentence (such as 'Initiator'). Adverb SIDs provide the space and time context of a verb SID. Attribute SIDs indicate special properties or relations amongst the sentence entities, whereas atom SIDs specialise in defining values that are bound to certain event instances (for instance 'Username'). Lastly, referent SIDs allow the linking of two or more parts of a sentence ('Refer to' and 'Refer as'). There are additional SID types, but the aforementioned ones are the most commonly employed.

One can clearly observe a structural hierarchy for forming complex sentences that also contributes to the semantic meaning. This semantic structure is inspired by the syntax of natural languages. A verb is always at the heart of every sentence and is followed by a sequence of one or more qualifiers that describe the various entities that play parts in the sentence, or qualify the verb itself.

```
(And
        (OpenApplicationSession
            (When
                (Time 14:57:36 24 Feb 2005)
            )
            (Initiator
                (HostName 'outside.firewall.com')
            )
            (Account
                (UserName 'tom')
                (RealName 'Tom Attacker')
                (HostName 'frigg.uio.no')
                (ReferAs 0x12345678)
            )
            (Receiver
                (StandardTCPPort 22)
            )
        )
        (Delete
            (World Unix)
            (When
                (Time 14:58:12 24 Feb 2005)
            )
            (Initiator
                (ReferTo 0x12345678)
            )
            (FileSource
                (HostName 'frigg.uio.no')
                (FullFileName '/etc/passwd')
            )
        )
        (OpenApplicationSession
            (World Unix)
            (Outcome
                (CIDFReturnCode failed)
                (Comment '/etc/passwd missing')
            )
            (When
                (Time 15:02:48 24 Feb 2005)
            )
            (Initiator
                (HostName 'hostb.uib.no')
            )
            (Account
                (UserName 'ksimpson')
                (RealName 'Karen Simpson')
                (HostName 'frigg.uio.no')
            )
            (Receiver
                (StandardTCPPort 22)
            )
        )
    )
```

Figure 2.4: CISL sentence syntax example

CISL [17] is not only about semantic rules. Its authors were concerned with the encapsulation of the structured semantic information into the 'Generalised Intrusion Detection Object' (GIDO), data structures that hold the encoded event information. The purpose of encoding the information in a standard way is to make the process of exchanging the information amongst various IDS components easy.

Unfortunately, despite the well-conceived interoperability target, the CISL GIDO encoding process introduced many problems. Doyle [19] has criticized many of the aspects of the CISL GIDO structure. Although the purpose of the document was to evaluate the fitness of CISL for use in the DARPA Cyber Command and Control (CC2) initiative, the paper identifies serious inadequacies that concern the CISL time resolution data representation facilities, as well as data throughput limitations caused by the fixed size of the GIDO data structure. Finally, Doyle comments on the lack of support for the next generation Internet Protocol (Version 6). Whilst these points are fair, they could easily be corrected by making the necessary changes to the relevant data types and overcome the perceived obstacles. In fact, section 7 of the CISL standard [17] contains specific guidelines that explain how to add information to a GIDO, to clarify or correct its contents. This suggests that the encoding principles are certainly extensible.

A more serious aspect of Doyle's critique [19] refers to the semantic structure of the CISL language. In particular, his criticism that CISL has "no facilities for representing trends or other complex behavioral patterns; ill-specified, inexpressive, and essentially meaningless facilities for representing decision-theoretic information about probabilities and utilities" indicates that the language would be a bad choice for describing threat prediction related information. The basic reasoning behind this critique is that CISL is too report-orientated and threat mitigation requires a different level of information, not just mere report structures of what is happening on one or more systems. These indeed represent more serious limitations that would require a more radical re-design of the CISL.

In addition to Doyle's criticisms, from a threat specification perspective, we note the following omissions/weaknesses in CISL:

- Inability to express variability in intrusion events: For example, all the necessary time patterns to specify recurring events of significance: The 'When', 'Time' and others SIDs can bind an event to an accurate time and date location. However, this is of little value to a threat specification as the accurate time of an intrusion event is rarely known. An SID operant such as 'afternoon-hours', 'evening-hours' would be more functional. This is true for other type of SIDs such as 'FileSource', network SIDs, etc. The expressions clearly lack the necessary polymorphism required to describe a range of possible events.
- The nesting of S-Expressions does not facilitate logical operands/operators, in order to describe alternative events. This affects the overall polymorphic description at event level.
- General lack of a mechanism to express confidence of a particular metric: Decision theoretic information is a desired feature of threat specification. The process of specifying a threat might include the description of various events. Not all of them have the same level of reliability and as such, a language that omits a mechanism of expressing a confidence in a particular event is a serious issue. This omission also limits the ability to build up user profiling information.

Nevertheless, Amoroso [20] characterizes CISL [17] and its associative Common Intrusion Detection Framework (CIDF) [21] as a “good piece of computer science”, despite the fact that it has not managed to infiltrate the IDS/IPS vendor market as a product interoperability platform. CISL is significant for the development of ITPSL for the following reasons:

- It's the first language framework for generic intrusion specification with system interoperability as its design goal, attempting to bridge the gap between language semantics and operating system/IDS product implementation details. This is a desirable feature because a good threat specification mechanism should focus on the threat itself and less on platform specific issues.
- It introduces the S-expression as a way to group the SIDs with the actual data in a hierarchical semantic notation which can nest expressions. Despite the previously discussed weaknesses of its proposed semantic identifiers, the suggested combination increases the clarity of the expression and the S-expression nesting capability increases the specificity of the statement in a consistent manner (the more S-expressions nested together in a the more specific the conditions of the match).

Panoptis [18] is another interesting and more recent intrusion specification language paradigm. The language sits on top of an anomaly detection system which parses standardized UNIX audit process logs. After establishing a user profile based on a number of different criteria, the audit logs are parsed and then checked against the profiling data. A sample of the entities and quantitative criteria that the panoptis system checks against is given below. These include:

- Discrete entities are organized in database tables such as:
    - tty UNIX terminals.
    - uid Users.
    - uidtty Users logged in on a specific terminal.
    - comm Commands.

  - uidcomm Users executing a specific command.

- Process accounting data such as:
  - maxaxsig Signal exit status.
  - maxhog Maximum CPU hog factor (CPU time over elapsed time).
  - maxmem Maximum memory usage.
  - maxavrw Maximum average disk block input/output.
  - maxstime Maximum system time.
  - minbmin Minimum daily start time (start time whithin the 24 hour interval).
  - maxutime Maximum user time.
  - maxbmin Maximum daily start time.
  - maxasu Superuser status.
  - maxcount Maximum number of times a given record has appeared in the database.
  - maxrw Maximum disk block input/output.
  - maxacore Core dump flag.
  - maxavio Maximum average character input/output.
  - maxafork Fork status.
  - maxetime Maximum clock time.
  - maxavmem Maximum average memory usage.
  - maxio Maximum character input/output.

In essence, panoptis is an anomaly detection system envisaged to detect a number of attacks such as data leakage, wiretapping and user masquerading amongst others. The semantics are restricted to configuration file options such as the one illustrated by Figure 2.5. Declarations of the type variable=value and a keyword(entity, value(s)) combination make up the syntactical convention.

```
HZ = 100            # "Floating point" value divisor
bigend = FALSE        # Set to TRUE for big endian (e.g. Sun), FALSE
              # forlittle endian (e.g. VAX, Intel x86)
map = TRUE          # Set to TRUE to map uid/tty numbers to names
EPSILON = 150        # New maxima difference threshold (%)
report = TRUE        # Set to TRUE to report new/updated entries
unlink = FALSE        # Set to TRUE to start fresh
# Reporting procedure
output = ’| /usr/bin/tee /dev/console | /bin/mail root’
# Databases and parameters to check
dbcheck(tty, minbmin, maxbmin, maxio, maxcount)   # Terminals
dbcheck(comm, ALL)                       # Commands
dbcheck(uid, ALL)                       # Users
dbcheck(uidtty, maxcount)                  # Users on a terminal
dbcheck(uidcomm, minbmin, maxbmin, maxutime,      # command
maxstime, maxmem, maxrw, maxcount, maxasu)
# Map users and terminals into groups
usermap(caduser, john, marry, jill)
usermap(admin, root, bin, uucp, mail, news)
```

Figure 2.5: A configuration file sample showing the DSL syntax of 'panoptis'

For instance, the statement dbcheck(tty, minbmin, maxbmin, maxio, maxcount) will check the UNIX terminal entity activity against the normal minimum and maximum startup time, as well as the maximum character input/output and the maximum number of times a given record has appeared in the database. If any of these figures exceeds the preset epsilon threshold normal value by 150% (EPSILON=150 declaration), the observation will be flagged as an intrusion. Note that these checks will be performed against the records of certain users as defined by the usermap statements (caduser, john, marry, jill as user group 1 and admin, root, bin, uucp, mail, news as user group 2).

The simplistic semantics of panoptis suffer from many of the previously discussed drawbacks of CISL. Development on the 'panoptis' system has been discontinued and thus, it is not fair to really judge the effort on the basis of the presented system. The panoptis authors were more interested to present a paradigm whose scope was to parse system audit logs and not a full intrusion specification language.

However, the panoptis approach is an important paradigm for an insider misuse specification language for two reasons:

- It is one of the first specification language approaches that target insider misuse incidents. The authors claim that under certain conditions, panoptis could “detect an employee transferring inordinately large amounts of data to a computer outside the organisation even if that employee had proper system authorisations to perform.”[18]
- It is one of the first systems that employs a Domain Specific Language approach, in order to design the intrusion specification semantics and capture precisely the domain's semantics. Chapter 7 will examine the Domain Specific Language more closely.

For all these reasons, both CISL [17] and Panoptis [18] play an important role for the development of insider misuse specification languages.

CIDF's work was continued by the IETF Intrusion Detection Working Group (IDWG). It addressed most of the CISL's GIDO encoding issues by introducing a new Object Oriented format for encoding and transmitting Intrusion Detection related information. The Intrusion Detection Message Exchange Format (IDMEF) [22] enriched the type of standardized information that IDS sensors may represent, as well as the process of standardizing the exchange of messages using protocols such as IDXP [23] and data exchange languages such as XML [24].

For example, the IDMEF “Confidence” and “Impact” classes can now be used to represent decision theoretic information. The earlier can assign a confidence and thus a probability to an observed event, whereas the latter relates the occurrence of a privilege escalation event to a numeric integer that indicates the seriousness of the event in a running system . This functionality can serve as the basis for encoding decision theoretic information.

The IDMEF standardization features were an important step towards facilitating intrusion and threat alert specification and ensuring better interoperability amongst Intrusion Detection System (IDS) products. The IDWG work has concluded without correcting and standardizing the semantic structure of the CISL language. The IDMEF draft [22] proposes encoding and data structures for alert information exchange, but it does not suggest semantic guidelines like the ones proposed by the CIDF framework. Consequently, for IDMEF, the term „language‟ refers to the data types and encoding principles for IDS alert data and not to the syntactical guidelines of an Intrusion Specification Language.

IDMEF's XML adoption as a mechanism to encode and validate IDS alert data is an important foundation that influences the design of the ITPSL semantics. Chapter 6 elaborates more on the pros and cons of using XML as the basis for encoding the ITPSL semantics.

At this point, this Chapter has identified the problem domain outlining the profile of a typical insider and discussed relevant intrusion specification research efforts, in order to provide the basis for specifying insider threats. Specification is an important step in the process of understanding the nature of a threat. If one is able to specify a problem, it can be inferred that the problem is sufficiently understood, in order to make the first step towards its mitigation. Despite the fundamental research paradigms presented by intrusion specification research efforts the field produced frameworks that were not interoperable and descriptive for insider threat specification.

The transition from intrusion to insider threat specification is a new research area and a core issue of IT misuse. A good understanding of what constitutes an insider threat is derived by the ability to systemically define the threat. Thus, the next Chapter makes the transition from generic threat specification to insider threat specification, introducing essentials concepts and misuse language paradigms.

2.4 Conclusions

This Chapter defined the IT misuse problem domain by means of providing essential definitions for the terms insider and misuse. Case studies and recent survey data were presented to prove that legitimate users can pose a serious threat for the health of IT infrastructures, either by means of intentional (malicious) actions or by accident. Finally, two relevant intrusion specification examples (CISL, Panoptis) were discussed, in order to set the scene for the concept of insider threat specification.

Chapter 3 Insider Threat Specification

3.1 Defining insider threat specification

Chapter 1 defined the term 'threat' [1], whereas the second chapter elaborated on the term insider [11]. Drawing upon these definitions, we must now define the term 'insider threat specification', as ITPSL is concerned with specifying insider threats.

Insider Threat Specification is the process of using a standardized vocabulary to describe in an abstract way how the aspects and behavior of an insider relate to a security policy defined misuse scenario.

The need for a standardized vocabulary is satisfied by the existence of taxonomies of the research domain, many of which are discussed in later paragraphs. These taxonomies form the foundation for designing suitable ITPSL semantics. The way they describe and abstract threat issues shapes what ITPSL should be describing.

Personality, organizational role, financial status and access credentials are some examples of insider aspects. In contrast, the insider behavior refers to the actions of an individual for accessing, representing or deciding about organizational assets.

A security policy defined misuse scenario implies the existence of a monitoring policy. Chapter 1 mentioned the security policy [10] that defines in plain language the borders between acceptable and unacceptable usage of IT resources. However, this plain language description must then be converted into suitable monitoring statements. Bace [25] discusses the difference between a security and a monitoring policy by providing the following example: Assume that an organization has the following line in its security policy:

'*Access to the patient financial information is restricted to the accounting clerk*.'

This should be translated to the following monitoring policy statement, representing an if-then rule-based pseudo-code of a system-level monitoring application:

'*If patient financial information is accessed and subject is not a member of the group "accounting-clk", then generate an alert*'.

An important observation to make in this case is that the transition from security to monitoring policy adds system-level specificity. From a generic statement, we go to a more specific one, expressing a policy in terms of the subject user (potential misuser), a user group (part of an IT infrastructure's authentication system) and the act of accessing the information.

This shift towards more IT system specific (yet still abstracted) parts and user actions is indicative of what a misuse detection and prediction language should describe. ITPSL is closer to the monitoring policy of an organization. A misuse monitoring policy examines user actions against misuse scenarios. Figure 3.1 illustrates the relations between user entities, the security and monitoring policies and the various components of the IT infrastructure.

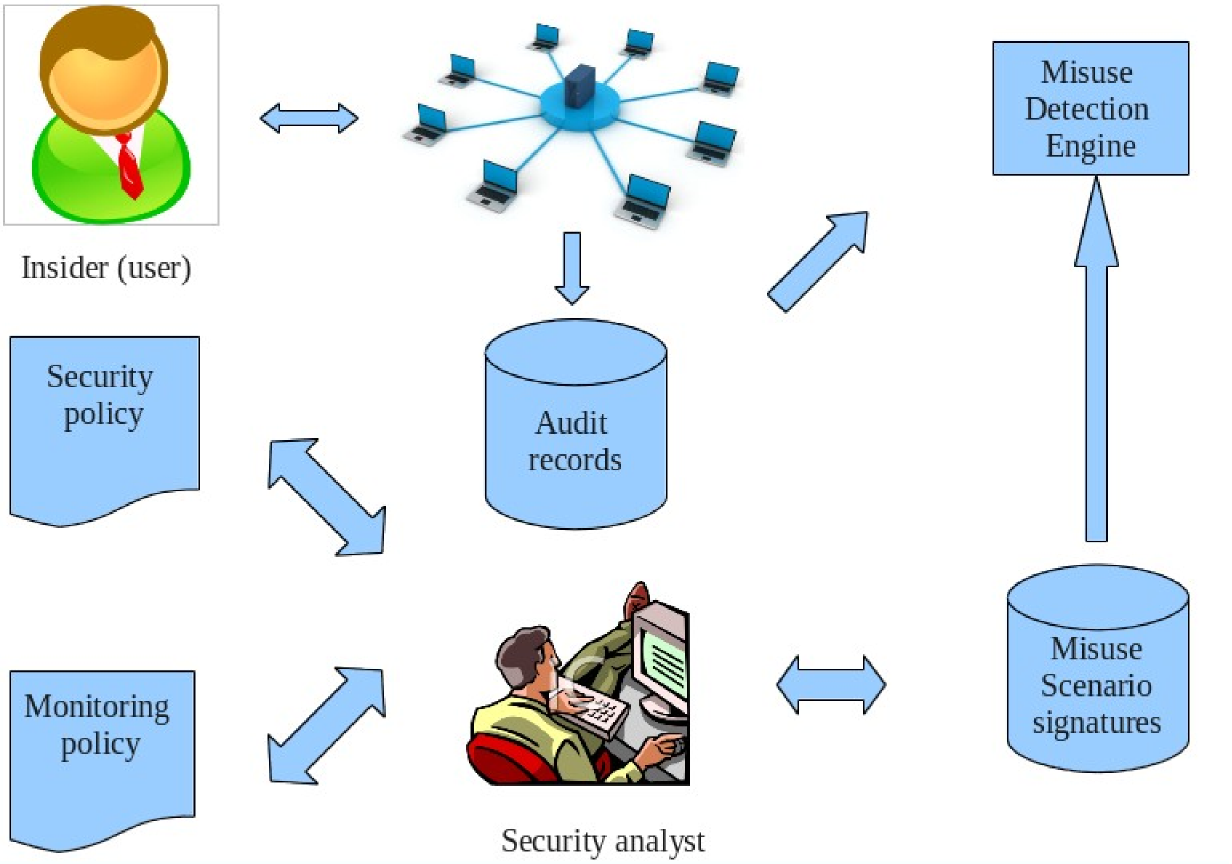


Figure 3.1: Misuse detection information flow

The security analyst translates the Security (and resulting monitoring policy) into a set of misuse scenario signatures. The involvement of a specialist is necessary as the process of translating from security to monitoring policy is error prone. A signature is a convention to encode a pattern of events that is known to be part of a misuse act. Misuse detection [25] employs pattern matching techniques expressed by signatures to detect known threats. In contrast, anomaly detection [25] is another major technique that facilitates detection of threats by means of sensing deviation from a normal behavior. Both techniques complement each other and it is possible to use signatures to employ both traditional misuse detection and anomaly detection criteria. This is demonstrated in later sections of the document that discuss the proposal of ITPSL semantics.

The misuse scenario signatures and collected audit data [25] from the IT infrastructure are fed into a misuse detection engine that infers the presence of threats. The presence of reliable and relevant audit data is very important, in order to make use of the produced misuse signatures. Chapter 5 provides an in-depth discussion of the audit data requirements for insider threat specification and proposes a suitable audit engine for that purpose.

Lastly, how the threat description relates to a misuse scenario provides two important contexts for insider threat specification:

- **The threat detection context**: Concerns detecting the occurrence of the threat scenario.
- **The threat prediction context**: Concerns detecting signs of the threat scenario.

A clear distinction between the two contexts is important for specifying insider threats. Threat detection should encompass all the essential events that constitute the threat scenario as part of the specification. On the other hand, a threat prediction context should include all the essential events plus relevant event precursors. The prediction process is of course concerned with assessing the relevance of the event precursors to a particular threat. This implies that insider threat specification in a prediction context should adequately describe both the precursor, as well as their relevance to what is being predicted. This is the role of decision theoretic information being incorporated inside the threat specification process. This gives ground to Doyle's criticisms [19] of CISL [17].

Another attribute of the distinction between the insider threat detection and prediction contexts concerns the ability to specify the temporal basis of events. The graph of Figure 3.2 below illustrates the concept. Imagine that an organization has an explicit rule that forbids the downloading of pirated material using peer-to-peer applications within the local IT infrastructure. As a result, a security analyst uses a specification mechanism (in threat prediction context) to tackle the threat. If the x-axis represents time, the significance of events (y-axis) increases the closer we get (time-wise) to the actual downloading of the pirated material. Each sub-block specifies a unique combination of events. If the user searches for P2P applications on a search engine (event 1), up to the point where he executes the installed P2P application, he is not breaking the rule. Hence, events 1 to 3 constitute the threat precursors and a clear security policy should distinguish between the precursors and the threat (event 4). However, the event correlation is time dependent (sequence of events) and hence temporal information is important for specifying threats.

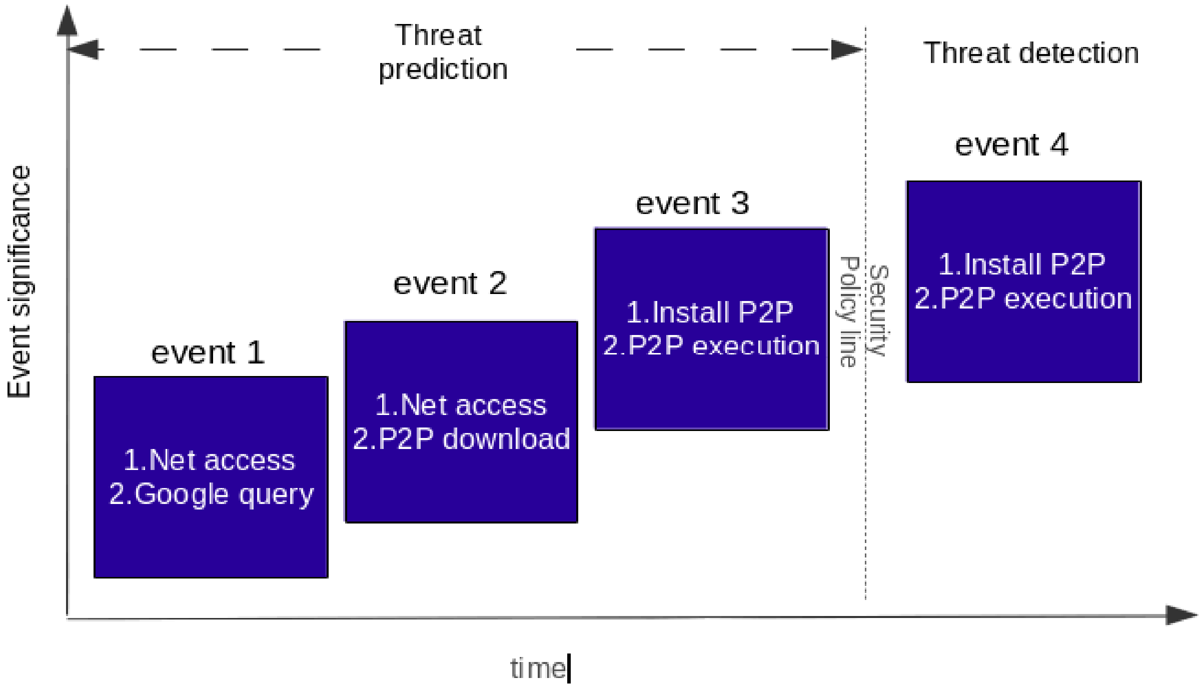


Figure 3.2: Temporal information in threat prediction context

3.2 An overview of misuse detection languages

The previous section concluded with a discussion of the importance of temporal information as part of the threat specification process. The temporal indications as well as the discrete events identified as part of a misuse signature (Figure 3.2) are at the heart of the design of misuse detection languages. In the context of information security, misuse detection languages are research tools that describe misuse activities in a consistent manner. Consequently, it is important to consider notable some examples of these tools and decide which of their design aspects can be borrowed by ITPSL.

The field of misuse detection has devised two types of languages:

- **Textual representation misuse detection languages**: These languages use purely text signatures to represent a misuse detection scenario.
- **Graphical representation misuse detection languages**: This type of misuse detection languages employs a range of state transition analysis [26] methods (Finite State Automata [27], Colored Petri Nets [28]) to model and study intuitively misuse scenarios.

Notable examples of textual representation misuse detection languages are the Production-Based Expert System Toolset (P-BEST) for the EMERALD project [29] and the Rule Based Sequence Evaluation Language (RUSSEL) for ASAX [30]. Both of these language paradigms are based on 'if-then' rules that are wrapped around semantics to infer attacks. The inference ability is important because it forms the basis to encode decision theoretic information, a desired property of insider threat specification.

P-BEST [29] is of particular importance, as it is one of the first systems that facilitates host and network based misuse detection by means of combining a 'modus ponens' inference schema with suitable system-level semantics. In formal logic, a 'modus ponens' (MP) assertion is a simple argument form by which we can infer q from p if : (p => q) and p we deduce q. In other words, if we have a set of events/facts that are part of a threat (threat detection context) or precursors of a threat (threat prediction), we can infer the threat by means of MP statements that are called production rules. The events/facts are often implemented by semantic attributes and their values represent the state of the domain we are examining.

Figure 3.3 below contains the semantics of a P-BEST production rule that infers a buffer overflow attack. The 'rule' keyword associates a block of conditions and an inferred event (buffer overrun attack) via the '==>' operator. Each of the conditions is the result of an Operating System audit record reduction operation ('AUE_EXEC' and 'AUE_EXECVE' are special audit record flags that mark the execution of a program, whereas the '^\\' string indicates some arguments of interest that are known to cause buffer overflow conditions for specific applications). We can also observe a semantic consistency: Each of the conditions is enclosed in square brackets and the entire production rule is contained within a square bracket block preceded by the 'rule' keyword.

```
     rule[BSM_LONG_SUID_EXEC(*):
          [+e:bsm_event]
          [?|e.header_event_type==’AUE_EXEC|| e.header_event_type==’AUE_EXECVE]
          [?|e.subject_euid != e.subject_ruid ]
          [?|contains (e.exec_args, "ˆ\\") == 1]
          [?|e.header_size > ’NORMAL_LENGTH] ==>
     [!|printf("ALERT: Buffer overrun attack on command %s\n", e.header_command)]
]
```

Figure 3.3: P-BEST production rule semantics to detect a buffer overflow

The P-BEST work is important not only for setting the foundations of expert systems in misuse detection. It was also one of the earlier examples of cross-platform misuse detection languages, as it spanned an entire range of research and development Operating Systems that were used at the time, from Free-BSD and Solaris to Linux. Moreover, section 8 of the P-BEST paper [29] provided one of the first acknowledgments of the need to create public misuse detection signature creation tools, by envisaging a web service that would allow users to create their own rule sets and apply them to their environment.

```
rule Masquerader()
 begin
 i f evt=‘login’    and    not habitual_term( userid, terminal)
               and    not habitual_time( userid, timestp)
     —>    Trigger off for next Watch(userid, profileof(userid) ) ;
 fi ;
 Trigger off for next Masquerader()
 end

 rule Watch ( suspectid: integer ; profile:table[cmnd:byte_string, maxtimes:int] )
 i f userid = suspectid and evt=‘login’
     and habitual_time( userid, timestp) and habitual_term( userid, terminal)
     —> skip;
     userid = suspectid and is_an_entry_in( evt, profile)
     —> begin maxtimes := select(evt,profile);
                 Trigger off for next
                   Count_rule3 (suspectid, maxtimes, timestp+ 3600, evt );
                 Trigger off for next Watch( suspectid, profile)
           end ;
     true
     —> Trigger off for next Watch( suspectid, profile)
 fi
```

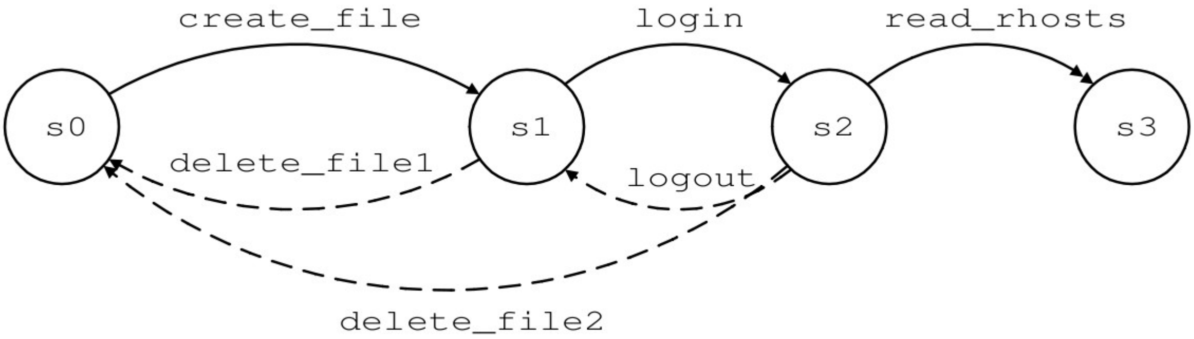
Figure 3.4: RUSSEL semantics to detect a masquerader

RUSSEL [30] maintains the 'if-then' structure but also provides additional programming constructs to specify and infer intrusive activities. These programming constructs include (beyond the conditional part of a production rule) structures to describe repetitive events, compound actions and logical operators to express event variance. This is an important aspect of the semantic structure of a threat specification, as it increases its expressive power. Figure 3.4 provides a RUSSEL example.

This RUSSEL example is concerned with detecting masqueraders, legitimate users that try to disguise in the system as a different person/user. Detection is performed by means of a compound action: A set of conditions on the 'Masquerade' rule triggers the detection of criteria described in a different rule ('Watch'). Apart from the trigger mechanism, one should take note of the logical operators (AND, NOT), as well as the fact that the layout resembles procedural programming languages ('begin' and 'end' keywords).

On the other hand, language designs such as the Event Description Language EDL [31], STATL [32], SUTHEK [33] and IDIOT [34] are examples of graphical representation misuse detection languages. Instead of using just a textual representation of the events that constitute a threat, a graphical representation can provide a more intuitive understanding of the threat scenario. This is because it provides a level of abstraction that separates the detection rule semantics from the actual algorithms [33], a desired property for insider threat modeling.

Figure 3.5 illustrates a state diagram and the associated STATL semantics to encode an ftp-write attack. In this attack (which is now addressed in most FTP server systems), a malicious user of the FTP service creates a .rhosts file in a world writeable directory. The created file contains the IP address of a remote host and can then be used to bypass the authentication system of the FTP server (a valid login name and password are no longer required by the FTP server). Thus, the malicious attacker can obtain illegal access to the FTP service.



```
Use ustat;
scenario ftp_write
{
        int user;  int pid;  int inode;
        initial state s0 { }
        transition create_file (s0 -> s1)
           nonconsuming
         { [WRITE w]: (w.euid != 0) &&
             (w.owner != w.ruid)
           { inode = w.inode; }
  }
```

```
State s1 { } transition login (s1 -> s2)
  nonconsuming
{  [EXEC e] : match_name(e.objname, "login")
  {  user = e.ruid; pid = e.pid; } }
state s2 { }
transition read_rhosts (s2 -> s3)
  consuming
{ [READ r] : (r.pid == pid) && (r.inode ==
inode) }
state s3  { {
  string username; userid2name(user, username);
  log("remote user %s gained local access”,
username); }}}
```

Figure 3.5: FTP write attack in STATL [32]

The state transition diagram (STD) of Figure 3.5 contains the discrete states of the attack. A state represents a snapshot of the system condition. Obviously, for the purposes of conceptualizing the misuse attack, we are not interested in representing the complete state of the operating system (file, network, kernel structures, etc) but only the events that are important for the attack to occur. State transitions (denoted by the arced arrows) represent an action that will change the system condition and help the attack evolve.

How the misuse attack evolves is the important thing to consider in a state transition diagram. Figure 3.5 illustrates three different types of arced arrows:

- Solid arc with single arrowhead: This type of transition indicates a non-consuming state transition. A non-consuming state transition is one that cannot prevent the execution of further instances of the misuse attack. In other words, both the source state and destination state for the transition are still valid.
- Solid arc with dual arrowhead: This type of transition indicates a consuming state transition. A consuming state transition prevents the execution of further instances of the misuse attack. The source state of the transition is no longer valid, as the consuming state moves the misuse one step further.
- Dashed arcs: An unwinding transition invalidates all events between the source and destination state, so it rolls the snapshot of the system back.

Considering the exact specification of misuse detection event instances is fundamental in the process of insider threat specification. If one is able to distinguish between different occurrences of a specified misuse detection attack, this means that they are able to correlate events and provide a more precise detection mechanism.

In the FTP write attack example of Figure 3.5, states s1 to s3 involve non consuming transitions. This means that there are multiple login processes on the FTP server, however, we are only interested in ones that read the .rhosts file after the login procedure is invoked (consuming state), indicating the evolution of the attack scenario. Alternatively, if the user deletes the .rhosts file prior logging into the server, that would roll the system back to an earlier snapshot, as the STD indicates.

The rest of the mentioned graphical misuse languages follow similar approaches and they model the state transitions using slightly different methodologies. EDL [31] uses Petri nets to study not only the state transitions, but also note pre and post conditions for each event described in the signature. In addition, EDL introduces the notion of signature re-use, by means of detecting signature similarities amongst a set of sequences. Like the authors of P-BEST [29], the EDL creators place emphasis on the creation of signature repositories. EDL emphasizes the importance of systemically producing accurate signature repositories by abstracting the relations amongst a set of signatures. In this case, abstraction means generalizing the signatures by taking away all the attack specific specifications.

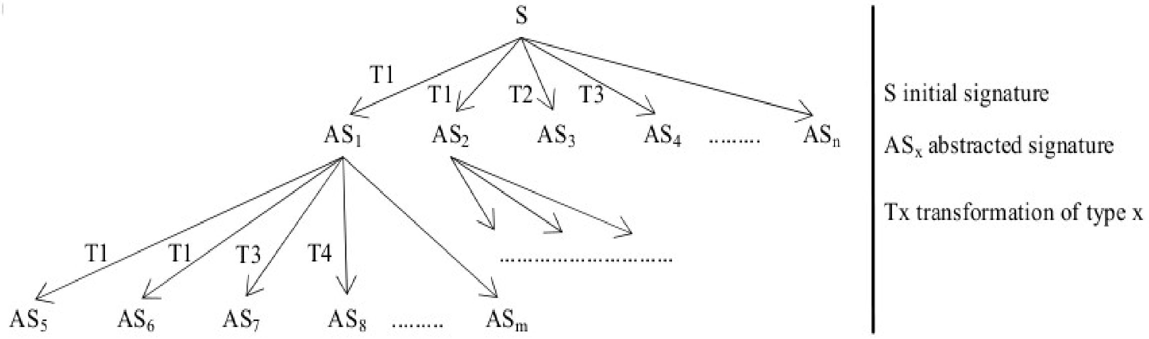


Figure 3.6: A signature abstraction tree (SAT) in EDL [31]

Figure 3.6 shows how the signature abstraction can be mapped by using a signature abstraction tree (SAT). Consider an initial signature S that is valid (carefully crafted by an expert analyst and verified to detect a known misuse incident). The objective is to produce valid signatures that detect variations of the initial misuse incident S, as well as valid signatures that detect more complex events. Starting from the root node S, we iteratively transform the original signature by discrete transformation processes T. For example, signature AS_1 is abstracted by signature S by means of transformation T1. AS_1 can then be further abstracted to produce another signature AS_5.

An abstracted signature associated to a node of the abstraction tree may detect all signature manifestations of the signature of the parent node. Thus, a SAT defines a set theory [35] inclusion operation over all nodes that represent attack manifestations. In particular, if one considers MS , MAS_1 and MAS_5 to be discrete manifestations of the misuse signatures S, AS_1 and AS_5 respectively, then the following set relation is true: $MS \subseteq MAS_1 \subseteq MAS_5$.

At this point, the previous discussion of misuse detection languages reveals notable characteristics that an insider misuse specification language should have:

- The semantics should facilitate a 'modus ponens' inference rule structure (if-then).
- It should express event timing information, describing either relative or absolute time stamps or events in sequence.
- It should enable the language user to focus on system-level specific events, describing file, network and process execution activities.
- The semantics should enable the language user to detect discrete manifestations of misuse incidents (one or more).
- Utilities to re-use signatures (repositories) should be available.

3.3 The semantic properties of misuse signatures

The last section provided an important wish-list of insider threat specification characteristics . This list was drawn by reviewing a number of existing misuse detection languages. However, the critical question is how to design semantics so that it is easier for analysts to devise valid signatures, avoiding scenarios of false negative and positive detection. Meier's seminal work on misuse languages semantics [36] is a useful guide that emphasizes the importance of the semantic design by stating: ”wrong understanding of the semantics of attack signatures is one of the main sources of false alerts or undetected attacks”. The same work also sets the foundations for facilitating the encoding of decision theoretic information in a semantic framework and provides a checklist of “must-have” validity and expressiveness features of a misuse detection language. Thus, it is worth considering it carefully in the following paragraphs.

Meier's work [36] re-iterates a number of points that have been covered in earlier paragraphs, such as the need for temporal event descriptors, the need to detect discrete manifestations (instances) of misuse incidents and the need to distinguish between consuming and non consuming steps. He defines a complex event as a set of interrelated simple atomic steps (as recorded by the audit system). The occurrence of the complex event is matched by the occurrence of one or more of the interrelated steps (which he calls basic events).

Meier's work goes into much more depth when it comes to systematizing the various levels of semantic event descriptions. In particular, the work defines three dimensions of semantic event descriptions:

- Event pattern dimension: This dimension examines how the event occurs examining aspects of:
    - Type and order: Basic events may occur in sequence, in variations (disjunction – logical OR operator), concurrently (conjunction – logical AND operator), or simultaneously (in parallel) as part of a complex event scenario. In some circumstances, one should also be able to express basic events that are not allowed to be part of the complex event (negation – logical NOT operator).
    - Repetition: This aspect facilitates the useful ability of specifying the number of times a basic step must occur in order to match the complex event (i.e. number of unsuccessful login attempts within the hour).
    - Continuity: If one considers a stream of consecutive events of type A,B and C, are we allowed to have an event instance c occurring between event instances a and b?

      Concurrency: When specifying more complex event patterns (complex events that contain other complex events), it is necessary to be able to state whether the constituting complex events may overlap or not overlap.
    - Context conditions: Sometimes it is necessary to specify constraints on the context in which a complex event occurs. This increases the language specificity. For instance, we might define a particular set of conditions inside a single event specification to a specific file, computer host or network connection. This is an example of an intra-event context condition. In direct contrast, an inter-event context condition is one that matches at least two events. For example, the same set of conditions could be applied to a specific file in two different computers (two matching events in computers x and y). Inter-event context conditions are the basis for event correlation.

The reader might assume at this point that some redundancy exists between the conjunctive and simultaneous operators of the 'type and order' aspect and those of the 'concurrency' aspect. This is not the case. The 'type and order' concurrency operators are used to express concurrency at the basic event level, whereas the ones of the 'concurrency' aspects regulate the overlapping of complex events. The combination of the two in a complex event pattern (complex event of complex events) creates different possible scenarios for event execution flow. Figure 3.7 illustrates the difference between the two types of concurrences.

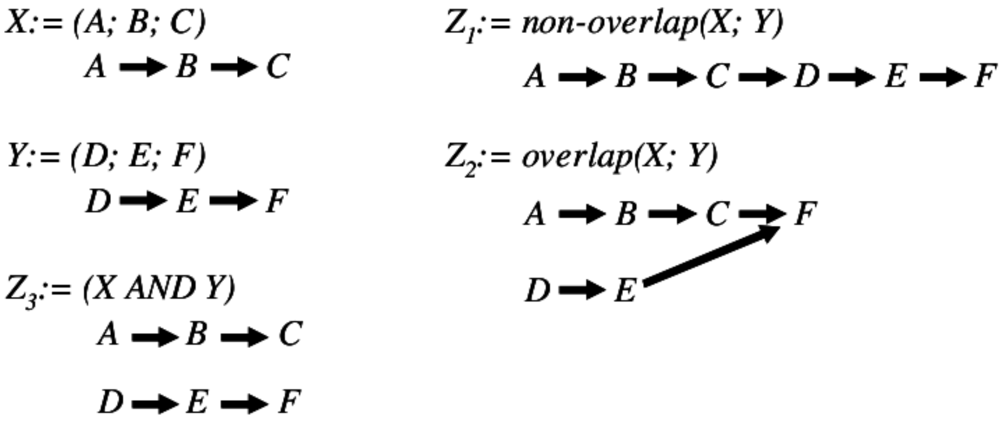


Figure 3.7: Event concurrency operators by Meier [36]

Let us consider the following complex events: X:=(A; B; C) , Y:=(D; E; F) Z1:=non-overlap (X; Y), Z2:=overlap(X; Y) and finally Z3:=(X AND Y). A non-overlapping sequence of complex events X and Y (Z1) prohibits any overlap between the events: if event D was found before event C, that would not match the pattern Z1. On the other hand, for an overlapping specification of the same complex event pattern (Z2), if events D and E occurred at the same time as A and B, but the entire sequence finished with F, the event pattern would match the event specification. Finally, if we applied the 'type and order' conjunction operator, what we really mean is that we have an execution...Overlapping specification is often considered the default specification.

- Step instance selection dimension: Instance selection has been discussed in earlier paragraphs, referencing the STATL [26] semantics. Consider an event type pattern E3:= (A; B; C) where events of type B correspond to log events that document the values of a network event e.g. the number of occurrences of a specific network endpoint. In addition, suppose the following audit trail t2 records the following event instances: t2:= {a1 b1 b2 b3 c1}. The question now is which instance of type B events shall be selected as a valid match for event type B. There are three possibilities:
  - Selecting the first instance: Often used as the default choice, this would validate an event instance pattern {a1,b1,c1} to match the event pattern E3.
  - Selecting the last instance: If we use the last instead of the first event instance, the following pattern {a1 b3 c1} will match event pattern E3.
  - Selecting all instances: This means the pattern {a1 b1 b2 b3 c1} would match the event pattern E3.

- Step instance consumption dimension: This final dimension is concerned with detailing the description of consuming and non-consuming events, as described in Figure 3.5 (FTP write attack example based on STATL [26]). In contrast to concurrency operators that concerns events, instance consumption concerns individual events. An event is consuming when its occurrence cannot create a partial event instance, and non-consuming when its occurrence can create a partial event instance. To illustrate the concept more formally, consider the event patterns E5:=(A; consuming B; C), E6:=(A; non-consuming B; C) and the audit trail t3:={a1,b1,a2,b2}:

- If we try to detect pattern E5, valid matches are the {a1,b1} and {a2,b2} subsets. This is because b1 consumes a1 and does not allow partial considerations of further occurrences of B in the event stream.
- If we try to detect patter E6, we have four valid subsets of t3: {a1,b1} , {a1,b2}, {a2,b1} and {a2,b2}.

Whether we choose to mark a step as consuming or non-consuming depends on its nature. For example, when describing events at system-level, a process fork [30] should normally be a non-consuming step, as any of the process children might be relevant to the misuse incident. On the other hand, exit events (destruction of a process or deletion of a file) are considered consuming steps.

Meier's dimensions [36] act as a good benchmark of the expressive and precise definition facilities of a misuse detection language. None of the previously discussed intrusion specification and misuse languages are complete in terms of fulfilling features like step instance selection and consumption. However the work does not propose specific semantics. His criteria determine the abstracted functional role of semantics. This is one important aspect of designing semantics. The other one is the selection of words and structures to express the context and domain of misuse detection.

Magklaras and Furnell [37] propose a methodology for deriving semantics for insider misuse detection and prediction that includes three important steps:

- the abstraction of the domain, which involves the removal of all the unnecessary details of the environment;
- the systematic categorization of the necessary (abstracted) details into language semantics;
- the process of engineering the developed semantics into software.

The domain abstraction is a critical part of the overall language design process and raises the question of which entities and data are relevant to the insider threat domain. Meier's abstraction [36] and the domain abstraction of [37] are examples of models. The following chapter will introduce the concept of insider threat models and will discuss further the ways of abstracting the insider threat domain.

One final note prior to closing this section should be devoted to the issue of specifying insider threat prediction. Figure 3.2 illustrated the temporal nature of insider threats. Moreover, earlier paragraphs discussed the issues of step selection and consumption as they were derived by graphical misuse detection languages. These two concepts provide the foundation for the partial detection of threats and the basis of insider threat prediction. However, none of the previously discussed misuse detection language efforts provide a mechanism for assessing the weight of each step, in a multi-step misuse detection scenario. This is a serious omission and later parts of this thesis discuss the necessity of such a mechanism, in order to predict threats.

3.4 Conclusions

In summary, this chapter defined the term insider threat specification and its two important contexts: the insider threat detection context, where the goal is to detect a particular type of threat, as well as the insider threat prediction one which aims to predict threats, both considered as insider threat mitigation measures. Relevant misuse detection language studies were also discussed, in order to identify important insider threat specification functional capabilities, such as the 'modus ponens' rule inference structure, the ability to express event timing information, the focus on expressing system-level events, the discrete event specification capability and the ability to re-use specification rules easily. The identification of these capabilities and their rigorous examination by Meier [36] creates the ground for drawing a set of functional specifications for designing an insider threat prediction and specification language. These issues are examined in great detail in the next Chapter.

Chapter 4 Insider threat taxonomies and models

The previous chapter presented the essence of the insider threat specification and prediction concepts, a number of misuse detection languages and concluded with a discussion of essential insider misuse signature properties. Hence, it provided one of the foundations of the proposed insider threat prediction and specification language (ITPSL). This chapter will provide an additional foundation by abstracting the problem domain, categorizing important semantic entities for ITPSL and listing the scope and functional requirements of our proposed language.

4.1 Insider misuse taxonomies

In Chapter 2, the CISL [17] and Panoptis [18] paradigms proposed certain types of data to monitor, without giving a concrete explanation on why these data were chosen and how they can aid the threat detection/prediction process. The first chapter of the thesis presented notable insider misuse case studies and surveys and concluded that whilst generic trends can be spotted, this is not enough information to have a concrete picture of the problem. In order to select with confidence a range of insider misuse threat descriptors, a more systemic view of the problem is needed. Information security taxonomies and threat models provide the answer to these questions.

Taxonomies are efforts to classify information. Threat Models are attempts to make use of the systemic knowledge of the taxonomies and estimate threat levels and/or simulate threat scenarios to help insider misuse researchers understand better important concepts and ultimately estimate threat levels. A model is an “abstraction of the system being studied rather than an alternative representation of that system” [38]. This abstracted representation of the system should closely resemble its real-world behavior. The process of abstracting a real-world situation implies that not all information about its attributes and functions is transferred into the model. Only those attributes and functions that are relevant for the study of certain aspects of the entities involved are included. Thus, insider threat model designers need to consider carefully which attributes and behavioral characteristics of a legitimate user are important to a threat estimation process. Later paragraphs of this chapter will discuss these issues in more detail.

As with intrusion specification languages, intrusion specification taxonomies are not a new idea in the information security field. An overview of intrusion specification taxonomies is provided by Furnell et al [39]. Amongst these taxonomies, one that specifically addresses insider IT misuse incidents is given by Tuglular [40]. This taxonomy integrates an established security policy to the process of classifying computer misuse incidents in three dimensions: incident, response and consequences. These dimensions can be divided into additional sub-dimensions that further classify a particular misfeasor. Tuglular's paper is one of the first to suggest a 'target-type of threat' association as a way to prevent insider misuse. The target is an 'asset' and the rule is called a 'strategy' in the terminology he proposes. The suggestion is mentioned in a single sentence and forms the basis for a methodology to predict insider misuse threats. If one can associate successfully certain actions to threats then it establishes the first step towards systematizing insider IT threat prediction.

Most research efforts in the field of intrusion taxonomy classification are still at an early stage. The Tuglular taxonomy, and others mentioned in [39], are useful for the systematic study of intrusions, but they offer little help to a process designed to automatically detect intrusive activities. This is because the classification criteria employed by these taxonomies cannot be qualified or quantified very easily by an Intrusion Detection System with the level of information they exhibit. Moreover, none of these taxonomies is tailored for the process of estimating the likelihood of Insider Threat.

The best way of enhancing the expressiveness of an intrusion taxonomy scheme for insider misuse activities is to focus on the human actions and how their consequences impact the elements of the IT infrastructure that are being targeted. The idea is that it is easier to detect which particular element is affected by a potentially intrusive action, rather than focusing on the task of sensing the motives for initializing an attack or focusing on other non-system detectable factors of the insider misuse domain.

The first thesis chapter demonstrated the perplexing variable nature of insider IT misuse. What is considered as misuse by a well-defined IT usage policy is not the same across different organizations. Hence, another important property of a suitable Insider IT misuse prediction taxonomy is the freedom of the security architect to choose what can be considered as an Insider IT misuse threat indicator and even decide on the confidence of each indicator. Most taxonomies enforce a rigid framework for classifying phenomena with clear borders of distinction that offer little space for subjective or varying interpretation of facts. This schema does not fit the case of Insider IT misuse prediction. As a result, one can construct a suitable threat prediction taxonomy based around consequences detected at system level [37]. Figure 4.1 below displays the top level of the taxonomy structure indicating the three primary, non-mutually exclusive levels that address these consequences.

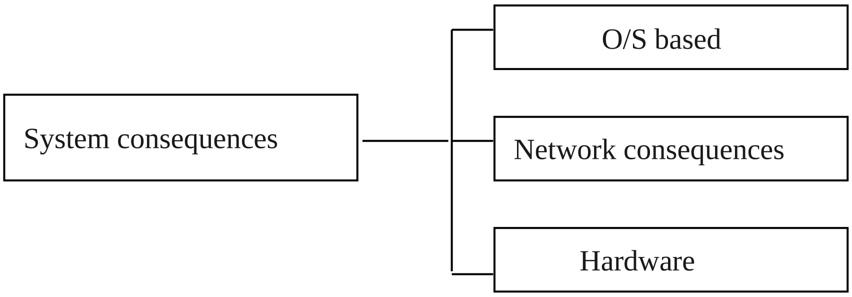


Figure 4.1: Top hierarchy level of an insider misuse taxonomy

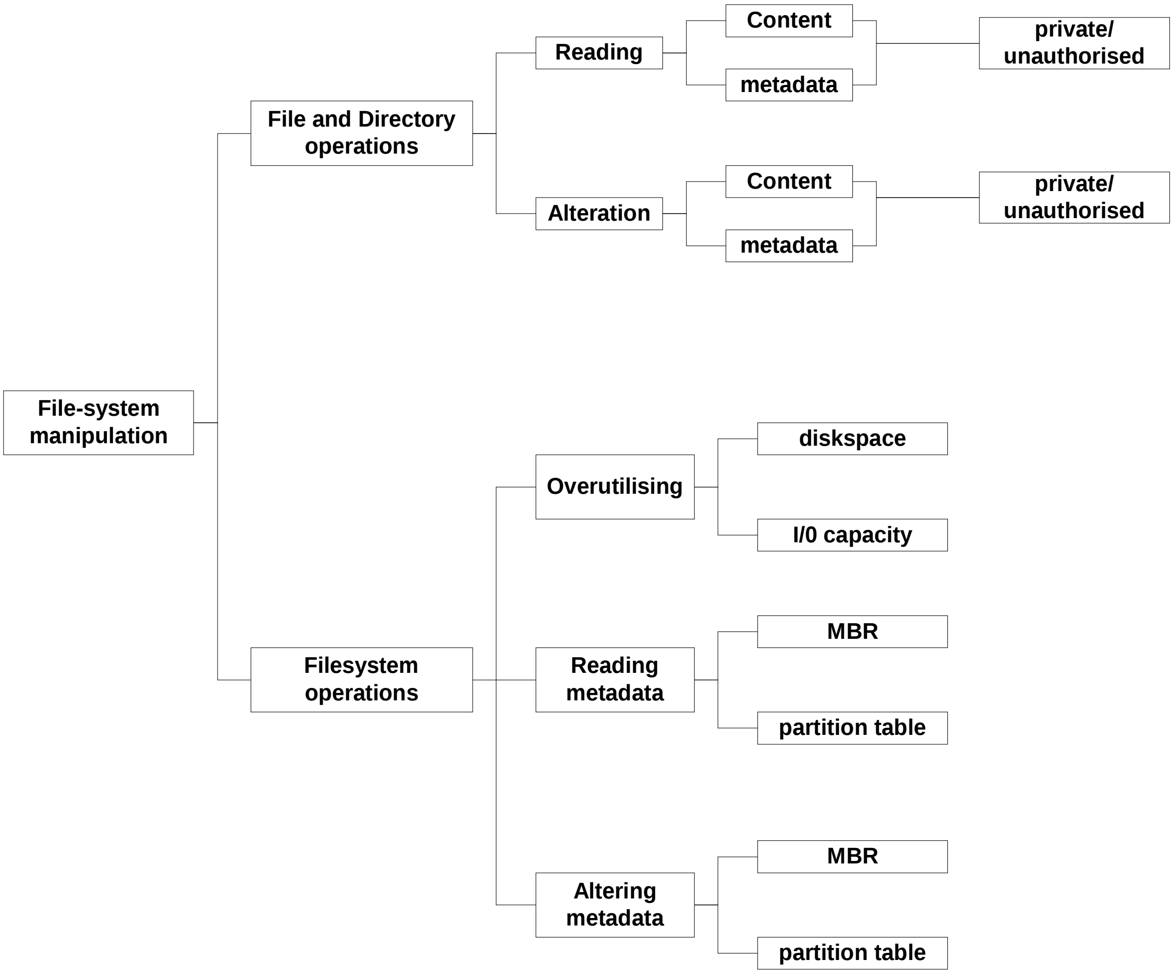


Figure 4.2: File-system manipulation O/S consequences

The Operating System (O/S) based consequences are branched down to two sub-levels of file-system and memory manipulation, illustrated by Figures 4.2 and 4.3 respectively. A justification for this is that a large number of security faults [41] involve filesystem and memory management issues, and indeed the core modules of UNIX [42] and Windows-based [43] operating systems provide (amongst others) specific support for the related functions. Hence, it is safe to assume that these two kernel functional attributes can be used as a strong criterion for further classifying legitimate user activities.

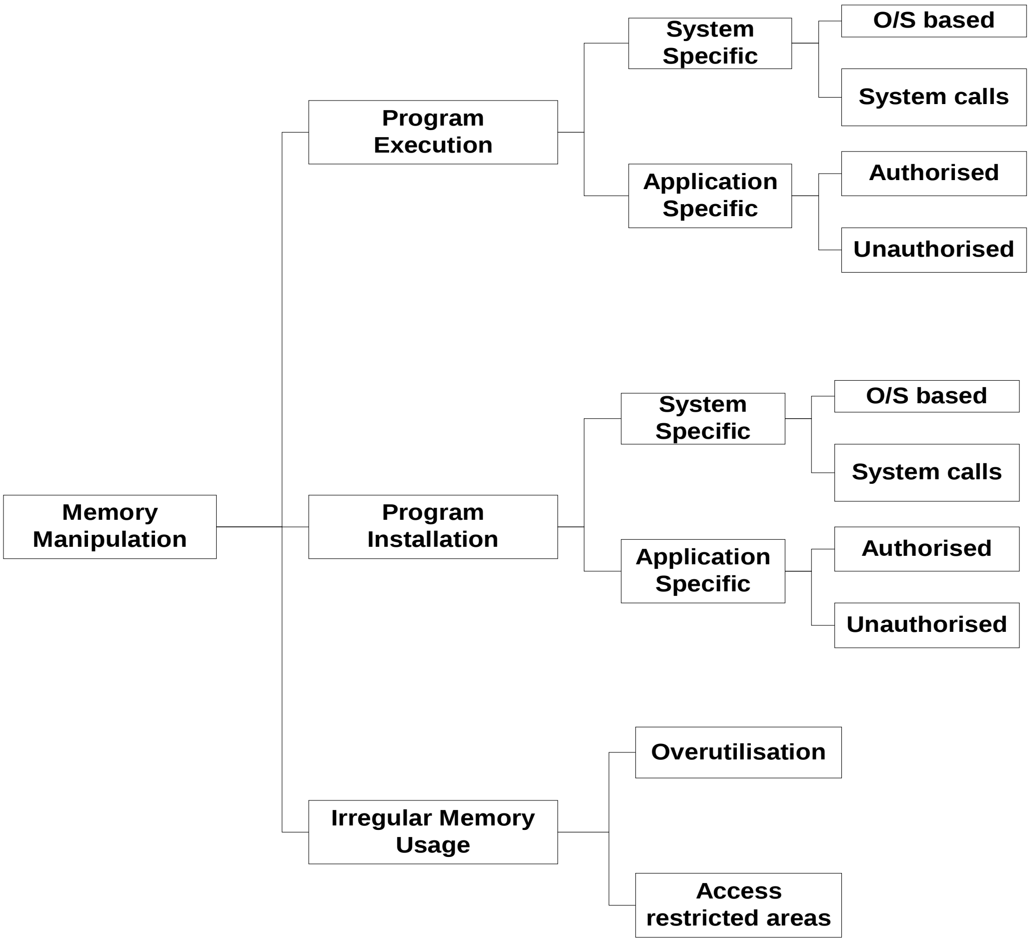


Figure 4.3: Memory manipulation O/S consequences

At File/Directory level, a misuser may attempt to read or alter (write/create) certain files. These files might contain sensitive or unauthorised information (information theft or fraudulent modification of vital information). A knowledgeable insider might also attempt to read or modify file information that is not directly related to its content. Bach [42] and Richter [43] emphasize that most Operating Systems allow a file to contain additional information such as access/creation/modification times as well as information that relates the file to its owner and permits access to it under certain conditions. Although the mechanisms that implement these file attributes are different amongst Operating Systems, they are collectively known as file metadata and they are vital mechanisms to secure the privacy, availability and integrity of the file contents. Consequently, they are good candidates for exploitation by a legitimate user who is about to perform a deliberate or accidental misuse act.

The points mentioned in the previous paragraph are also valid for 'filesystem' related data. Every Operating System organizes its files and directories by means of a specific set of rules that define how a file (contents and metadata) are about to be stored on the physical medium. The Operating System sub-modules that handle these issues are known as filesystems. Attempts to read or alter the physical medium's Master Boot Record (MBR), intentional or accidental modification of partition table data are some of the most notable auditable actions that could point to legitimate user misuse acts. Chapter 1 mentioned Robert Hanssen's attempt to hide information in modified floppy disks [12]. This is a classic reminder of this kind of activity.

In addition to filesystem content and metadata modification, a small-scale survey of insider misuse conducted by Magklaras and Furnell [44] showed that excessive disk space consumption is perceived as a problem for many of the respondents. Under certain conditions that depend on the configuration of the IT infrastructure, a legitimate user might produce a deliberate or accidental Denial of Service attack (DoS).

Memory inspection is the best way to see if a legitimate user attempts to run or even install a suspicious program. Indeed, it is one of the core techniques used in the detection of overtly malicious code, such as viruses and Trojan horse programs. The usage of unauthorised programs is a serious issue that can also create a way for accidental misuse by introducing a number of system vulnerabilities, as described by Papadaki et al [45]. The execution or installation of these programs could be intercepted by either recognising a program's footprint in memory or by intercepting a well-known series of system calls produced by various suspicious programs. For example, the fact that a non-advanced user is trying to compile an advanced vulnerability scanning tool is an event that should be noticed and serve as a good indicator of potential misuse activities that are about to follow.

In addition, attempts to consume large memory portions of an operational system that are related to a legitimate user account can serve as good indicators of (intentional or accidental) insider misuse at Operating System level. One might argue that the 'irregular memory usage' sub-categories should really belong under the 'Program execution' hierarchy of events. However, it is possible that someone will produce a quick and easy Denial of Service attack on a running system by forcing the host to commit large portions of system memory to a process, as demonstrated in various case studies described by Moore et al [46]. Moreover, a large category of security faults can be achieved by means of accessing normally restricted memory areas, creating what is commonly known as a "buffer overflow" attack [47]. As a result of these issues, it was felt that a separate sub-category hierarchy should exist to describe these events (Figure 4.3).

The filesystem and memory manipulation consequences conclude the O/S consequence category of the proposed taxonomy (Figure 4.1). The next category, “network consequences”, represents another distinct set of factors that could be taken into consideration in order to classify insider misuse threat indicators. Figure 4.4 illustrates the network-related consequences of acts that could be used as legitimate user threat indicators.

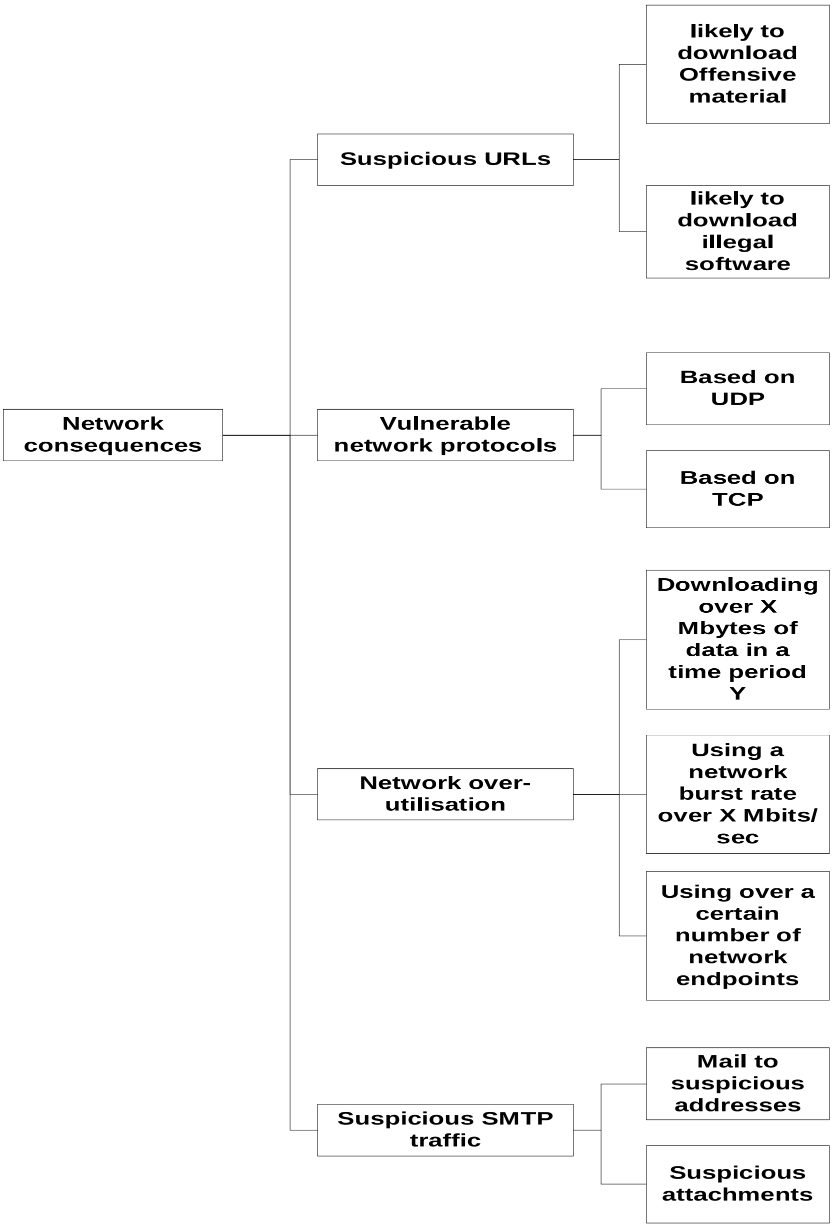

Network consequences
Suspicious URLs
likely to download Offensive material
likely to download illegal software
Vulnerable network protocols
Based on UDP
Based on TCP
Network over-utilisation
Downloading over X Mbytes of data in a time period Y
Using a network burst rate over X Mbits/sec
Using over a certain number of network endpoints
Suspicious SMTP traffic
Mail to suspicious addresses
Suspicious attachments

Figure 4.4: Network consequences of the insider IT misuse prediction taxonomy

Network packets that are associated with certain legitimate users and indicate the usage of a variety of network protocols and applications that might introduce certain vulnerabilities are also distinct ways of accidental or intentional IT misuse. For example, it could be said that a user that employs the TELNET [48] protocol to login to a multi-user system is more likely to have her account compromised than a user who logins via the Secure Shell (SSH) application [49] due to the fact that the earlier application transmits the user password in clear-text form across the network, whereas the latter one encrypts it.

Someone might also like to differentiate between TCP and UDP based applications/protocols. From a potential threat point of view, UDP services are less secure than TCP based ones. For example, Ziegler [50] discusses in detail how UDP's lack of flow control and state mechanisms can create various data security problems. Consequently, the distinction between the usage of UDP and TCP services can serve as a potential insider misuse threat indicator, on the basis that UDP services are more likely to be accidentally (or intentionally) abused by a legitimate user.

Although the 'Filesystem Manipulation' subcategory of the taxonomy indicates ways with which disk storage capacity can be misused, the results of over-utilization can also affect network capacity. For instance, a legitimate user could start downloading massive quantities of data, exceeding the network bandwidth cost budget of a business (Downloading over X Mbytes of data in a period Y). The X and Y number limits can be selected by the network administrator according to the company budget requirements.

In addition, a legitimate user might also cause network congestion by exceeding the data network’s ‘burst’ or throughput capacity or exhausting the number of available network endpoints, as described by Sharda [51]. Bandwidth hungry applications, such as video streaming players, and multiple data transfers can cause congestion that can severely impact the performance of a data network or affect the Quality of Service (QoS) of certain applications that require sustained data network throughput.

Finally, incoming or outgoing SMTP headers or attachments might indicate activity related to e-mail misuse that can certainly be traced in network or host level. Outgoing e-mails that contain a set of particular files as attachments (e.g. password database files, other sensitive material) and have unusual destination addresses (e.g. unknown Hotmail accounts, a large number of recipients) should serve not necessarily as intrusion indicators but as insider threat estimators. The plethora of malicious code efforts and phishing techniques may have an external origin, but the threat is realized by the actions of unsuspecting legitimate users. In addition, proprietary information theft could also be realized by means of emailing sensitive material to non-authorized external entities.

The last system consequences category (“hardware”) plays an important role in preventing a number of computer system threats. Insiders can often access the physical hardware of the machine very easily. Thus, removal or addition of hardware components, as well as modifications of their default configuration are some of the events that may act as important indicators of insider misuse prediction in a computer system.

At this point, we have a taxonomy tailored to the needs of automated Insider Threat detection and prediction because:

- It is heavily based on factors that are easily qualified/detected at system level.
- It is flexible enough to allow the security architect to define what is considered as a threat element. For example, he could define which user network protocols are more likely to pose a threat to the system when they are utilised by a particular legitimate user. This is a necessary requirement because what can be considered as legitimate user misuse varies amongst different organisations.

However, in order to make use of these threat indicators, we need a way to quantify them. This paves the way for the discussion of various Insider Threat Models presented in the following paragraphs.

Earlier paragraphs defined the notion of a model as an “abstraction of the system being studied rather than an alternative representation of that system”. Consequently, the first important step of deriving an Insider Threat Prediction Model is to decide which attributes and behavioural (functional) characteristics of a legitimate user are important to the Threat Estimation Process. This will produce a set of Insider Threat Qualification Attributes (ITQAs). The whole process is discussed in the next section of this chapter.

4.2 Insider Threat Models

The next step in the process of establishing our proposed language is to look into insider modeling. An insider threat model's purpose is to describe how the ITQAs can be quantified, in order to estimate the level of insider threat per individual user. This will involve the establishment of a suitable mathematical function, which will take as input a number of ITQAs and will associate them with a certain level of threat. We shall call this function the Estimated Potential Threat function, which quantifies the ITQAs. At this point, the overall target of our ideal model will be achieved: the establishment of a mechanism that will map ITQAs to certain threat levels.

The development of insider threat models is a relatively new idea. Wood [52] provides an excellent basis for qualifying a set of metrics to mitigate insider threat. Most of these criteria are in line with the conclusions discussed as part of the previously described insider misuse taxonomy by Magklaras and Furnell [37].

In particular, Wood suggests that a malicious insider can be qualified in terms of distinct attributes:

- **Access**: The insider has unlimited access to some part or all parts of the IT infrastructure and the ability to physically access the equipment hardware. Consequently, the insider can initiate an attack without triggering traditional system security defenses.
- **Knowledge**: The legitimate user is familiar with some or all the internal workings of the target systems or has the ability to obtain that knowledge without arousing suspicion.
- **Privileges**: The malicious insider should not have problems obtaining the privileges required to mount an infrastructure attack.
- **Skills**: The knowledgeable insider will always have the skills to mount an attack that is usually limited to systems that he/she is very familiar with. The model assumes that a given adversary is unlikely to attack unfamiliar targets.
- **Tactics**: This attribute refers to the methods used to launch the malicious attack. They are dependent on the goal of the attack and might include a variety of scenarios such as plant-

hit-and-run, attack-and-eventually run, attack-until-caught as well as passive information extraction acts.

- **Motivation**: Insiders might launch the attack for profit or sabotaging the target organization. Some of them might mount an attack for personal reasons such as taking revenge against the enterprise or even satisfy their plans to invoke some policy change inside an organization.
- **Process**: The model assumes that a legitimate user follows a basic predictable process to mount an attack that consists of distinct stages. First the malicious adversary will become motivated to mount the attack. The next logical stages involve the identification of the target, the planning of the attack and finally the act of mounting the attack itself.

All of the previously mentioned attributes emphasize important aspects of the insider misuse problem. The discussed insider case studies of Chapter 1 have presented comments on the importance of insider attributes such as role, knowledge and privileges. A very useful comment with respect to the Insider Threat Estimation modeling comes from the process attribute. The fact that Wood characterizes an insider attack as a 'predictable' process is a positive sign for the goal of establishing an insider threat model.

However, Wood's criteria do not necessarily represent a clear picture for the establishment of an insider threat prediction model. Not all stages of an insider attack can be safely predicted. Some of the previously mentioned attributes are difficult to qualify by an Intrusion Detection System. The 'motivation' adversary attribute is one of them.

It is very difficult to establish a set of sensors that could reliably deduce when an individual becomes motivated to misuse a system. For instance, let us suppose that IDS sensors record that a commercially important file is transferred from a disk to an external storage medium in the early morning hours. The fact that this particular file transfer took place could be related to a malicious act or an innocent file backup process performed by the system administrator as part of a system recovery process. It is important to maintain a record of these types of events, but their existence does not necessarily indicate an insider misuse event in progress. The plethora of the potential origins of such an event would increase the amount of information to be evaluated. Consequently, the complexity of the algorithms to capture and evaluate this type of information would deem this attribute's exploitation impractical. At the time of writing, there is not a known algorithm, which is able to capture and evaluate that kind of information in existing Intrusion Detection Systems.

If someone observes the different stages of the 'process' insider-modeling attribute, it becomes clear that the closer we get to the actual attack itself, the stronger the indicators of insider threat. Although detecting motivation might be tricky, with a carefully chosen quantification scheme of ITQAs, someone could sense an adversary during the target identification and attack planning stages. This strategy goes along the line of thinking of our proposed misused taxonomy being based on system-level factors.

In addition, other attributes seem to be so closely related that might be redundant. For instance, it would be more logical to combine the attributes of 'access' and 'privileges' into one 'insider access rights'. The issue of obtaining a privilege to mount an attack should include logical and physical means of interacting with the systems. The same could be said for the attributes of 'knowledge' and 'skills', because the ways in which a legitimate user gets to know a system and what can be inferred from the insider's system knowledge are issues that are closely interrelated.

Due to its introductory scope, Wood's paper [52] does not deal with the quantification of insider threat attributes. It is unknown whether this means that a suitable threat modeling function has been deduced as part of the preliminary model mentioned in this paper. The author has yet to publish a completed version of the model for verification.

A more recent research effort by Schultz [53] presents a preliminary framework for understanding and predicting insider attacks by providing a combination of behavioural and system usage ITQA metrics. The paper mentions the detection of system usage patterns that may act as "signatures" of a legitimate user or certain indicators of an attack preparation ("deliberate markers" and "preparatory behaviour"). Legitimate users might also make noticeable mistakes in the process of misusing a system (meaningful errors). Finally, "correlated usage patterns" refers to sequences of actions that might not be detected in individual systems but they could certainly indicate misuse when considered against multiple systems.

Schultz also suggests that certain aspects of a legitimate user's personality could serve as threat indicators. In particular, on-line (e-mail, IRC or other forms of computerised human-to-human communication) verbal behaviour with signs of aggression, dominance towards particular people might serve as a good prognosis factor of certain attacks ("verbal behaviour"). Furthermore, based on the works of Shaw et al [54], the research suggests that it is possible to examine other "personality traits" as potential threat indicators.

The Schultz preliminary framework even suggests a way to quantify all these metrics by means of a multiple regression equation that consists of the summation of the ITQA metric variables multiplied by their weightings. If X_1, X_2, X_3… X_N represent the quantified ITQA metrics, Wi (i=1, i=N) their respective weights and C an arithmetic offset constant, then the expected estimated threat Xe is derived in Figure 4.5 below.

$$\mathbf{Xe = (\Sigma\ WiXi) + C = W_1.X_1 + W_2X_2 + W_3X_3 + \ldots + W_NX_N + C}$$

Figure 4.5: Schultz threat model equation

One notable absence of the Schultz insider threat prediction scheme is that there is no direct association between the estimated level of threat and the legitimate user's level of technical knowledge. Although the proposed metrics can provide evidence that could be used to infer the level of user sophistication, there is no mentioning of a mechanism that takes that into consideration. Given the fact that, at the time of writing, the field of Insider Threat modeling is premature to reveal any usable results, it is difficult to prove the real impact of user sophistication on the threat level. On the other hand, Wood's model, a number of case studies and the survey results (chapter 1) provide strong indications that there is a direct relationship between these two concepts. In that sense, the lack of a legitimate user sophistication gauging component could present a serious omission of the Schultz framework.

In addition, the exploitation of future mechanisms that will associate personality traits to potential misuse threat levels raises certain ethical and feasibility concerns. It is outside the scope of the thesis to examine ethical issues and the various laws that are associated with them. Nevertheless, the process of designing a model that is going to be employed in the real world should take into consideration its troublesome aspects. A metric that penalizes real people in terms of their character traits will be considered unethical by many and depending on regional legislation may be also not feasible to implement.

In summary, the Schultz framework is more refined than Wood’s earlier Insider Threat model in that it provides more concrete examples of ITQA metrics as well as a basic quantification mechanism for them. However the framework is still in its infancy. The author acknowledges that the chosen metrics need further refinement in order to prove their usefulness in a threat estimation process.

Both models concentrate on malicious (i.e. intentional activities) without considering accidental insider misuse actions. This can be a serious omission for a model that aims to address all aspects of the insider threat issue, as the problem of accidental insider misuse does exist and can have serious consequences (as shown in the Norwich Union versus Western Provident Association case [15], previously mentioned in Chapter 1).

Brancik's [55] seminal work on the insider threat modeling should be referenced as a good source of information. Brancik's efforts center around information alteration, which is an important element of insider fraud, despite the fact that insider misuse surveys indicate that the frequency of these incidents are lower than other most common misuse incidents (web and email abuse). His Tailored Risk Integrated Process (TRIP) is the most important contribution. However, a risk management process deviates from traditional modeling approaches. This is because it focuses on factor evaluation. Detection of threat metrics is not addressed extensively.

Finally, all of the aforementioned research efforts do not address the issue of managing the representation of the data that feed the model component functions. One could argue that a preliminary model design needs to focus more on the scope, quality and quantity of its insider threat modeling functions. On the other hand, a well-thought definition of the procedures that represent and store the data that feed the threat modeling functions may have a notable impact on the computational efficiency and acceptance of the model. The reasons that support the need for this requirement are going to become apparent in the following paragraphs.

For all these reasons, a more formalized and broader model description is needed. An Insider Threat Prediction Model that attempts to overcome the shortcomings of previous research work has been published by Magklaras and Furnell [56].

Considering a legitimate user population that has access to various components of an IT infrastructure, the core of the Insider Threat Prediction Model is a three-level hierarchy of mathematical functions evaluated in a bottom-up approach. At the top level, the Evaluated Potential Threat (EPT) function provides an integer value that quantifies and classifies the potential threat for each legitimate user into three different categories. If x denotes the computed EPT for a legitimate user, EPT_MAX a threshold EPT value for considering the user a threat and EPT_MIN a threshold EPT value for considering the user's on line presence as suspicious, then:

- Important internal threat (x ≥ EPT_MAX): It indicates a high potential of a particular user misusing the system.
- Suspicious (EPT_MIN ≤ x < EPT_MAX): This flags a condition where a particular user behaves in a manner that does not constitute a substantial threat but it is still a concern.
- Harmless (0 ≤ x < EPT_MIN): To indicate that the potential of misuse is nearly non existent for a particular user.

It should be emphasized that the derived EPT value is an integer that represents a measure of the likelihood of system misuse, ranging from 0 to 100 points. Higher EPT scores indicate more probable threats. However, it should be noted that the model equations presented in this Chapter do not represent a validated probabilistic model. Since EPT represents likelihood of Insider IT misuse occurrence, one would expect the formulae to map a series of data to a probability figure. Although this is the aim of the model, in addition to the EPT function, one would then have to carefully relate the derived EPT score to the fact of whether the event really occurred or not. This comparison would facilitate the construction of proper probability distribution function, which relates a range of data to a probabilistic value of incident occurrence. As a result, the reader should be aware that there is a difference between the EPT score and an actual probabilistic figure.

Each of the threat component functions models particular aspects of insider attributes and behavior. At the moment, in order to devise a well structured organization of threat components, the suggestion is to provide two threat component functions. The first one considers legitimate user attributes such as access rights and professional role, whereas the second evaluates potential threat simply by examining aspects of user behavior at the system level, as shown in Figure 4.6.

$$\mathbf{EPT} = \mathbf{F}_{\mathbf{ITPQA}} = \mathbf{F}_{\mathbf{attributes}} + \mathbf{F}_{\mathbf{behavior}} \Rightarrow$$

$$\mathbf{EPT} = \mathbf{C}_{\mathbf{role}} + \mathbf{F}_{\mathbf{accessrights}} + \mathbf{F}_{\mathbf{behavior}} \Rightarrow$$

$$\mathbf{EPT} = \mathbf{C}_{\mathbf{role}} + \mathbf{C}_{\mathbf{sysadm}} + \mathbf{C}_{\mathbf{criticalfiles}} + \mathbf{C}_{\mathbf{utilities}} + \mathbf{C}_{\mathbf{physicalaccess}} + \mathbf{F}_{\mathbf{sophistication}} + \mathbf{F}_{\mathbf{fileops}} + \mathbf{F}_{\mathbf{netops}} + \mathbf{F}_{\mathbf{execops}}$$

Figure 4.6: The Magklaras and Furnell model equation

It is envisaged that $F_{behavior}$ has a greater weight in the process of calculating the user EPT than $F_{attributes}$. Legitimate user attributes are important and should always be taken into consideration. However, it is expected that amongst two users that have the same attributes, it is the gauging of their behavioral characteristics that can decide which one is more likely to constitute a greater level of threat for the system. Hence, a total of 30 points will be contributed to EPT by $F_{attributes}$ and 70 points by $F_{behavior}$. $F_{behavior}$ is expanded in a number of sub-functions, namely $F_{fileops}$, F_{netops} and $F_{execops}$, corresponding to file, network and process execution operations in agreement with the proposed insider threat taxonomy described in Section 4.2.

In addition, Table 1 lists the maximum weights of the nine top-level EPT formula components that are explained in detail in later sections of this chapter. Some of these components are constants (C_{role}, C_{sysadm}…etc) that belong to the $F_{attributes}$ function, whereas others constitute sub-functions of the $F_{behavior}$ function that address the assessment of the legitimate user on-line behavior.

The sum of the weights adds up to 100. This corresponds to a probability range of 0%-100%. The derivation of the defaults maximum values is a consequence of the aforementioned ratio between $F_{attributes}$ and $F_{behavior}$. Due to the initial choice of weights between the $F_{attributes}$ and $F_{behavior}$ functions, the 5 constants of $F_{attributes}$ have a maximum score of 6 points, contributing a total score of 30. The rest of the EPT components, should total a score of 70 points attributed to $F_{behavior}$. $F_{sophistication}$ attributes 10 of these 70 points and the rest of the sub-functions can score a maximum of 20 points each. Consequently, the default values preserve that ratio and attribute almost equal weights for each sub-function component.

It should be emphasized that the proposed maximum weights on table 1 are not meant to be fixed. A system administrator/security specialist can re-define the maximum weights, in order to reward a particular metric that he trusts more than the others. For this reason, the nine weights of Table 1 constitute the Weight Matrix, a very important parameter for the ITPM system. The Weight Matrix allows a specialist to further tune the sensitivity of the model, depending on the way he constructs misuse signatures, his confidence on the various metrics and the nature of the incident he is trying to predict. The security specialist would have to tune the weights of the various metrics by means of trial and error, based on good knowledge of the impact one or more IT misuse actions would have on the system. This feature enhances the adaptability of the proposed model scheme and enables to represent decision theoretic information.

EPT Component	**Maximum Weight**	**Meaning**
Crole	6	The documented role of the user inside the organization
Csysadm	6	User has access to Operating System administration utilities
Ccriticalfiles	6	User has read/write access to commercially sensitive files
Cutilities	6	User is able to execute application critical utilities
Cphysicalaccess	6	User has physical access to critical parts of the IT infrastructure
Fsophistication	10	Rating of computer system user knowledge
Ffileops	20	Presence of signs of forthcoming insider misuse at

		file-level
Fnetops	20	Presence of signs of forthcoming insider misuse at data network level
Fexecops	20	Presence of signs of forthcoming insider misuse at program execution level

Table 1: A sample Weight Matrix in the Magklaras and Furnell model

The reader can refer to chapter 6 the Magklaras MPhil thesis [9] for more details of the model and the reasoning behind the design of the proposed threat estimation functions. Two important things from this model should be emphasized here:

- the inclusion of various ITQAs at various levels (file, network, process execution) to represent a variety of system detectable user events.
- The introduced Weight Matrix concept as a mechanism of expressing different levels of confidence for the various ITQAs for a particular threat description.

Both of these things play a great role in the design of the ITPSL. The next section will explore the ITPSL relationship to the threat model process, as well as the overall scope of its inception.

4.3 The scope and functional requirements of ITPSL

Information security surveys and notable insider misuse cases reported by mass media were discussed in chapter 1 of this thesis. In addition, earlier sections of the present chapter provided a detailed presentation of the insider misuse domain by introducing a suitable insider taxonomy and a resulting insider threat model. However, how a threat model fits to a threat description language is not very clear. Understanding the relationship between the two is vital for setting the scope of our proposed language. Figure 4.7 illustrates the relationship of the ITPSL and the proposed ITPM model.

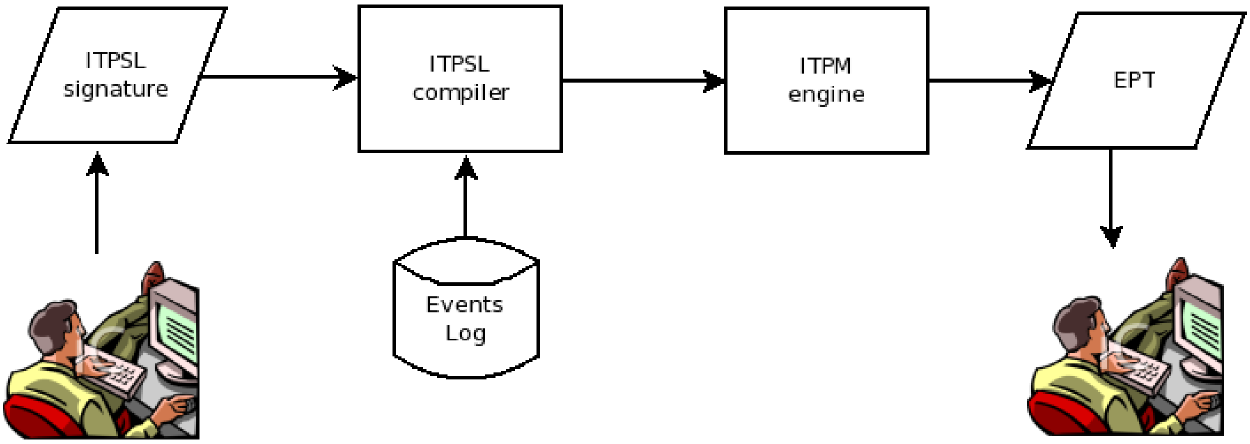


Figure 4.7: The relationship between ITPSL and a threat model

The flow of information starts with a security analyst writing a description of the particular insider misuse scenario, using the ITPSL semantics. The signature is validated by a compiler that translates the signature directives to query commands and makes use of an event logging infrastructure, in order to examine whether the ITQAs the signature mentions exist in the system. Apart from the semantics that qualify/quantify the ITQAs, the signature embodies a Weight Matrix statement which indicates the confidence of each specified ITQA. The results are passed to the ITPM engine which then derived an EPT value, indicating that a likelihood of a particular threat.

Figure 4.7 also includes the security analyst/system specialist both at the beginning of the information flow (misuse signature construction) and at the final stage, where the final assessment is done. This emphasizes that the analyst is in charge of the process, both in terms of defining what constitutes a threat and also in terms of judging whether the likelihood expressed by the model is accurate.

This places the foundation of the context of ITPSL as a component of an entire Insider Threat management architecture, as discussed in Chapter 7 of the Magklaras MPhil thesis [9]. The model estimates a threat which is described by a language and the likelihood is judged by the IT specialist. The emphasis is on the description and thus, the language addresses the lack of case repositories that express details of insider misuse incidents is apparent. An early report outlining aspects of the insider threat to the US government information systems published by the NSITSSAM Committee [57] considers the absence of case repositories as one of the limiting factors in the field of insider IT misuse mitigation research. In addition, the Carnegie Mellon University CyLab's 'Common Sense Guide to Prevention and Detection of Insider Threats' publication [58] states clearly the need to keep detailed records of employee actions in relation to file access, application usage and network connection matters.

ITPSL could also be a tool for digital forensic investigators. Digital forensics is an important research discipline of the information security field that is concerned with providing evidence to legal proceedings by means of gathering data to determine exact details of various types (internal and external origin) of system attacks. Brancik [55] mentions the importance of suitable tools to produce Key Fraud Signatures (KFS) to aid insider threat mitigation and thus signifies the overlap between insider misuse and the field of digital forensics.

The most widely used form of digital forensic investigation is quiescent or static analysis. For such type of analysis, an investigator would utilize a number of toolkits to make a forensically valid copy of the affected system's non-volatile data storage media and perform a “post-mortem” examination of the copied media. The goal is to examine static data (documents, images, email and system files) for digital evidence. AccessData's Forensic Toolkit [59] and Guidance Software's Encase [60] are two well known toolkits that perform, amongst other things, static digital forensic analysis.

However, static digital forensic analysis reveals an incomplete picture of the system in question. It cannot portray accurately the non-quiescent (dynamic) state of the system under investigation. Information such as active network endpoints, running processes, encryption keys for decrypted on-disk content, user interaction data (number of open applications per user, exact commands), as well as the content of memory resident processes may not be recorded accurately on non-volatile media. Hay et al. [61] discuss the shortcomings of static digital forensics analysis in detail. In order to overcome the barriers of static analysis, Adelstein [62] discusses the virtues of non-quiescent or live analysis, which essentially gathers data while the system under-investigation is operational. Microsoft's Computer Online Forensic Evidence Extractor (COFEE) [63] and FATkit [64] are two examples of tools that are able to extract live forensic data from volatile storage locations of a computer system.

Live data forensics analysis fills the gap of static examination methods, but it has its own disadvantages. Carrier [65] and Hay et al [61] discuss the risks associated with acquiring live digital forensic data. In particular, live analysis methods suffer from three basic problems:

- Investigator privileges: The investigator needs administration or escalated privileges to run the live analysis utilities. This could present a number of problems in environments where access policies prohibit escalated privileges from external entities to computer systems.
- System and data integrity:The data gathered during the live analysis phase might be compromised due to system (due to rootkit infection, misconfiguration or intentional alteration of data by one or more system users). Whilst the memory data retrieval issue has been addressed by some complex hardware configurations whose purpose is to reliably acquire volatile data from system memory, the rest of the data acquisition issues are serious and they stem from the fact that data are logged on the system under investigation and not on a safer area before they are analyzed.
- The “observer effect”: When static analysis methods are used, the investigator can examine the data without affecting the source media state. Unfortunately, that is not true for live analysis where the investigator's actions can affect the data. One would have to separate carefully the implications of the investigator's actions from the original data.

The previously mentioned needs in addition to the previously discussed desirable characteristics of Chapter 3 (Sections 3.2 and 3.3) shape the scope of the Insider Threat Prediction Specification Language (ITPSL): A specialized language that is able to encode system level data that concern legitimate user actions, in order to aid the process of misuse threat prediction and assist computer forensic officers in the process of examining insider misuse incidents. As such, ITPSL’s target audience is the security analyst/expert, as well as the seasoned IT administrator in charge of system operation and security issues. Both of these types of domain experts should be able to express insider misuse scenarios by using the language semantics to construct signatures of threat scenarios.

More specifically, the ITPSL language should be able to meet the following high level functional requirements:

- **FR1**:Store and analyze the data away from the target system(s) to minimize issues with maliciously or accidentally altering/deleting the data.
- **FR2**:The architecture of the language should facilitate the creation of suitable insider threat signature repositories, so that security specialists/system administration could easily browse for signatures of various threat scenarios. This feature aims to address the lack of suitable case repositories discussed in the earlier paragraphs of this section. The need for facilitating signature repositories was first discussed in section 3.2 of the third thesis chapter, when we reviewed a number of misuse detection language paradigms.
- **FR3**:Its semantics and logging mechanisms should facilitate the description of both static and live forensic insider misuse system data at the network, process and filesystem layer, in response to the issues discussed by Hay et al [61].
- **FR4**:The semantic description of user actions should satisfy Meier's event description dimensions [36], as discussed in section 3.2 of the third thesis chapter. In summary, the language should be able to:
    - encompass temporal indicators so that sequences of events could clearly be expressed and logged.

    - Its semantics should identify event step instance selection by being able to select a single or multiple occurrences of events and event steps.
    - Finally, ITPSL semantics should help the language user express step instance completion, by listing the desired first, last or intermediate occurrences of relevant events.
- **FR5**:The language should be able to represent decision theoretic information to address the criticisms of earlier intrusion specification examples such as CISL [17]. This implies the ability to consistently express various potentials scenarios of insider actions, giving the signature polymorphic properties.
- **FR6**:The semantics of the language should offer a consistent hierarchical way of describing a variety of scenarios and should be easily readable by humans and software modules.
- **FR7**: ITPSL should have an operating system agnostic scope. The signature author should use the same semantics to express the various misuse threat scenarios regardless of whether the underlying operating system is Microsoft Windows, Linux/Unix, MACOSX or other applicable platform. The language semantics should bridge any gaps created by operating system esoteric peculiarities that could affect the process of expressing threat indicators.
- **FR8**: The language semantics should also facilitate event correlation across multiple monitored systems. An insider threat scenario might be realized by means of involving multiple components of an IT infrastructure.

The previously listed functional requirements that were deducted from previous discussions serve as a guide for the start assembling the ITPSL building blocks. The next chapter presents one of the most important functional blocks of the proposed language: the audit engine.

4.4 Conclusions

This Chapter discussed the role of taxonomies in insider IT misuse research and presented an insider IT misuse taxonomy tailored to the needs of automated misuse detection and prediction. It also presented relevant misuse modeling efforts. A combination of a suitable taxonomy and model can form the basis of a language for detecting and predicting insider threats. Beyond the tasks of detection and prediction, the language semantics should also describe data in a complementary way to existing forensic tools, expressing static and dynamic computer forensic analysis data, in order to provide strong accountability of user actions. A detailed breakdown of the functional requirements of such a language concluded the chapter.

Chapter 5 The LUARM audit engine

Section 2.3 placed the origin of threat specification to the area of Intrusion Detection/Prevention System (IDS/IPS) [25]. Moreover, Figure 3.1 illustrated the information flow in a misuse detection language. One of the important building blocks of that flow was that of the audit record storage and this is the subject of this Chapter. In particular, the discussion examines the vital relationship between the audit record format and misuse detection language expressiveness. This will be followed by a presentation of existing audit record engines and a critique of them in terms of their suitability for insider threat specification. The chapter ends with a proposed logging engine that provides structured audit records made specifically for specifying insider threats, according to the functional requirements of ITPSL.

5.1 Audit record and insider threat specification expressiveness

Bace [25] discusses intrusion detection (and hence misuse detection) as an audit reduction problem. Audit reduction is the process of filtering the relevant information out of the audit records, in order to infer a partially or fully realized threat and excluding information that is irrelevant or redundant. In the process of designing an insider threat specification language, a great deal of emphasis should be placed on the content and structure of audit records, as they constitute the source of information of every misuse detection system.

The following paragraphs show that the expressive power of a misuse detection language is based on the wealth of information encapsulated in the audit record. The balance between too little and too much information on the audit record is a difficult one. Providing too much information makes the task of audit reduction difficult and not scalable, as the number of monitored system grows. In addition, redundant or irrelevant information might be difficult to relate to language semantics. In contrast, collecting data at a too coarse a level of detail can exclude vital information for the presence of a threat, cause false negative assessments and reduce the expressiveness of a misuse detection language.

The structure of an audit record is also important for a misuse detection language. A good structure has well defined fields that can be easily parsed and subsequently matched to data structures that represent language semantics. Moreover, the structure of the audit record should easily facilitate relational type [66] queries. This is because it is necessary for the information to be applied on the disjunction (OR), conjunction (AND), and negation (NOT) operators, as dictated by Meier's [36] event pattern dimension descriptors (Section 3.2) and ITPSL functional requirement FR4 (Section 4.3).

A final desired aspect of the audit record format is derived by the ITPSL nature. In essence, ITPSL describes insider actions. This means that the audit record should easily provide user entity accountability for each recorded action. Hence, a system action such as a file access should map clearly to a user entity. This desired mapping means that each recorded action could also be correlated to other actions of the same user, so that a set of actions can be related to a threat and the query language has enough information to perform step instance selection. Figure 5.1 illustrates such a correlation, by showing parts of the process execution and network endpoint creation log of a hypothetical audit record engine.

Let us assume that we wish to find whether user 'toma' has accessed the website 'www.suspicious.org', via a web browser, between 13:40 and 14:00 hours on the 25th of September 2010. In order to find such an event, we need to intercept the launching of a web browser process by user entity 'toma'. We would assume that the web browser will generate a network connection to 'www.suspicious.org'. The way one can relate the two events as part of a complex event (step instance selection) is to match the two events against a set of common identifiers, such as the process ID (PID), parent process ID (PPID) and username. This assumes that the process ID launching record has both PID and PPID data inside each process execution record. It also assumes that the corresponding network endpoint log structure has a corresponding PID and username field that could be correlated.

```
Process execution logs:
......
Logrec 158: /bin/bash, 13:55:16, 25/09/2010, 24208, 4288, toma
Logrec 159: /usr/bin/mapview, 13:55:18, 25/09/2010, 24209, 19504, nickb
Logrec 160: /usr/bin/firefox, 13:55:19, 25/09/2010, 38416, 2614,toma
Logrec 161: /usr/bin/firefox, 13:55:20, 25/09/2010, 24210, 24208, toma
Logrec 162: /bin/bash, 13:56:04, 25/09/2010, 43210, 8208, katbz
......
Network endpoint logs:
......
Logrec 256: localhost:43455, www.bbc1.co.uk:80, 13:23:01, 25/09/2010 ,34671, nickb
Logrec 257: www.bbc1.co.uk:80, localhost:43455, 13:23:02, 25/09/2010 ,34671, nickb
Logrec 258: mysql1.internal.domain:3346, localhost:5000, 13:38:14, 25/09/2010 ,14200, mysql
Logrec 259: localhost:8210, www.google.com:443, 13:55:20, 25/09/2010 ,38416,toma
Logrec 260: localhost:20310,www.suspicious.org:80, 13:55:21, 25/09/2010 ,24210,toma
Logrec 261:
......
```

Figure 5.1: Log event correlation and step instance selection

The log snapshot of Figure 5.1 shows that there are two instances of user 'toma' executing a web browser. Only one of them is relevant and the correlation can be performed because the PID and username are recorded in the network endpoint and the process execution audit records. In particular, only PID 24210 has connected to www.suspicious.org, which was started by a shell process (PPID 24208) of user 'toma'. The wealth and replication of vital information in various types of audit records is a requirement for proper event correlation and step instance selection.

Another important issue of audit record engines is that of referencing time. In large IT infrastructures that span several networks and time zones, audited systems might report in different time formats. They can also face 'clock skew' [25], a difference in time recorded amongst computer systems due to computer clock hardware inaccuracies, especially when an NTP [67] server is not available to provide a reliable time source. Clock skew is common amongst mobile components of the IT infrastructure, as well as amongst operating systems that run in virtual mode [68]. An audit record engine should resolve that problem and make sure that every record is entered into the log set by having a correct time stamp.

Finally, audit record engines should provide a scalable storage system to keep a large number of audit records available for future reference. Modern IT environments that consist of a large number of multi-user serving devices of different kinds can easily produce a large amount of data. If the stored information is consolidated to a single place, a natural choice for data availability and correlation, the amount of data can quickly overwhelm traditional file based storage approaches.

5.2 Existing audit record engines

Audit record engines have been around for a long time, since the very early days of operating systems. The following paragraphs will review a number of existing audit record engine specifications and solutions. The goal is to show that they do not fit all the requirements of misuse detection engines, as discussed in the previous paragraphs.

One of the earliest and most commonly referenced works that concern the format of audit records is the US Government's Trusted Computer System Evaluation Criteria (TCSEC – 'Orange Book') [69].

This was a structured evaluation process (Trust Levels) that specified the features and assurances required for operating systems and application software to contain and process classified information. As part of these assurance features, the 'Orange Book' specified extensive lists of events that audit systems should monitor. However, these lists were provided without guidelines for selection, so that an analyst could abstract what is being monitored and choose a set of them. Moreover, the 'Orange Book' audit requirements did not provide any specification for the structure and storage of audit records.

These omissions, as well as the age of the drawn requirements led to the cancellation of the Orange Book [69] by the US Department of Defense. The work is now purely a historic evidence of the need to draw audit requirements for operating systems. Instead, the Common Criteria for Information Technology Security Evaluation (CC) [70] standards have taken over the Orange Book's role. The CC effort does not fully address the previously mentioned audit record requirement omissions of its predecessor, the Orange Book. Despite enjoying an impressive industry product certification scheme and some criticism over the feasibility of implementing the listed requirements due to complexity [71], the CC effort has still to produce a comprehensive array of audit requirements. In comparison to the Orange Book, the CC provide a more structured audit functional requirement list, but still, no substantial discussions with regards to the content, format and storage of audit records.

However, we do take note of some of the high level functional audit requirements of their 321 page document [70]. In particular, CC requirement 88 of section 8.2 [70] states that: “At FAU_GEN.2 User identity association, the TSF shall associate auditable events to individual user identities.” In CC terminology TSF stands for Target of evaluation Security Functionality, meaning essentially the software and hardware under evaluation . This particular requirement we have already discussed with reference to step instance selection (Figure 5.1).

The CC effort [70] also states the minimum requirements for the content of an audit record by stating in requirement FAU_GEN.1.2: “The TSF shall record within each audit record at least the following information: a)Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; ...”. This is also in-line with the previously discussed issues about user accountability and temporal information. The outcome of the event might be a tricky to implement, depending on the context of the event. For some types of events that are atomic (i.e., an attempt to execute a file), logging success or failure is meaningful (i.e. to log that an attempt was made to execute a file might be an interesting fact) and feasible (this can be easily performed by monitoring for exit codes or testing for the execution of the program by using the userid credentials) . For other types of events that are more complex and concern many intermediate steps (binary program that performs many actions that do not always follow the same order/execution path) this is less trivial to implement, as it requires tapping to the actual system calls level or other proprietary application logs.

In addition to the minimum requirements for audit record content, the CC effort [70] defines desired ways of searching audit data by stating in requirement FAU_SEL.1.1: ”The TSF shall be able to select the set of events to be audited from the set of all auditable events based on the following attributes: a)[selection: object identity, user identity, subject identity, host identity, event type] b [assignment: list of additional attributes that audit selectivity is based upon] ”. This CC functional requirement implies that audit records need to be structured in ways that speed up or make easy the retrieval of data by different criteria.

Finally, the CC document [70] lists some notable high level functional requirements for the storage of audit records. The first one concerns the maintenance of the users that have access to the audit data (requirement 106, part of FAU_SAR.1): “maintenance (deletion, modification, addition) of the group of users with read access right to the audit records. ” Setting up a proper access controls policy for the audit data is a vital issue. The high-level approach of this requirement does not specify how the access control should be implemented. However, one can assume that isolating the credentials of the audit reviewers from the normal user authentication system is a useful step. This measure could be an important deterrent for accidental or intentional leakage of valuable audit data to ordinary users. It could also make the management (and activity logging) of audit data reviewers easier.

The second CC requirement that concerns audit record storage is that of FAU_STG (section 8.6) [70]. Actually this is a set of requirements that concern various aspects of the audit record storage. We quote from the requirements text:

“At FAU_STG.1 Protected audit trail storage, requirements are placed on the audit trail. It will be protected from unauthorised deletion and/or modification. FAU_STG.2 Guarantees of audit data availability, specifies the guarantees that the TSF maintains over the audit data given the occurrence of an undesired condition. FAU_STG.3 Action in case of possible audit data loss, specifies actions to be taken if a threshold on the audit trail is exceeded. FAU_STG.4 Prevention of audit data loss, specifies actions in case the audit trail is full. “. Once again, the requirements are given in high-level terms, specifying that:

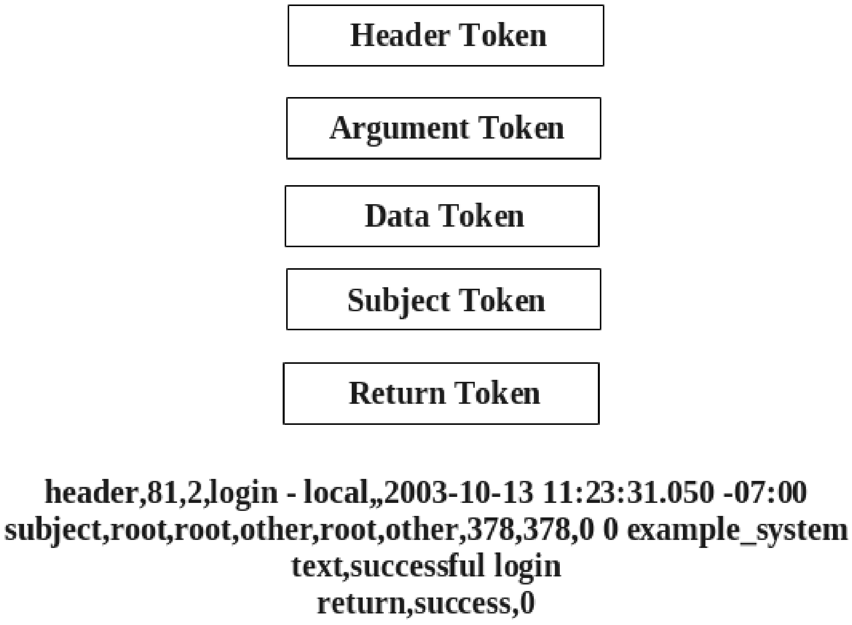
- unauthorized deletion and/or modification of audit records

- any other condition that could cause storage failure.

should be mitigated.

The point behind the Orange Book [69] and CC [70] is that they are both specifications and not implementations of audit record engines. However, several audit record engine implementations can be found in the literature.

The most common variety of audit record engines uses information that comes directly from the Operating System. Characteristic examples of this category of engines are Oracle's Basic Security Module (BSM) auditing system [72] and its open source implementation OpenBSM [73], the psacct audit package [74], as well as the syslogd [75] and WinSyslogd [76] applications (the latter runs on Windows operating systems).



```
header,81,2,login - local,,2003-10-13 11:23:31.050 -07:00
subject,root,root,other,root,other,378,378,0 0 example_system
text,successful login
return,success,0
```

Figure 5.2: BSM audit record format and example

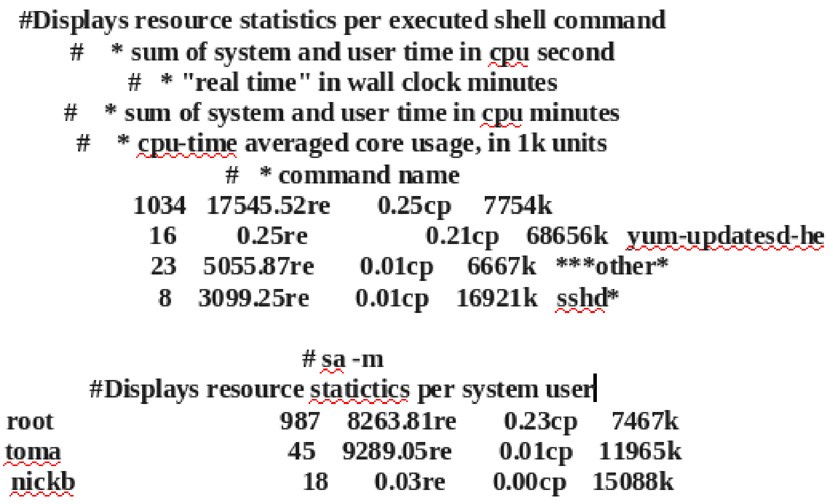
The BSM audit system [72] [73] has seen widespread deployment in commercial server grade operating systems. It structures its audit records in binary (non human plain text readable) files. Audit trail management commands are then used to decode the binary form of these files and produce human readable output.

Each BSM audit record is a series of byte encoded tokens. Figure 5.2 shows a typical structure of a BSM audit record and a corresponding decoded plain text example of an audited successful login entry. Actual audit records might vary in terms of the type and order of tokens. The Header token marks the start of the audit record. Argument and Data tokens normally contain data about the command and the arguments that caused an event. The Subject token states which process triggered the generation of the audit record. Finally, the Return token contains values that are returned by the process execution and can help the audit reviewer determine the success or failure of a command (the reader might recall CC [70] requirement FAU_GEN.1.2).

The OpenBSM initiative [73] has similar audit record structure with minor differences in the encoding of the different token types.

Pssact [74] is another audit system that generates operating system based audit trails. Although psacct can be used for security purposes (system administrators can check login attempts and user activity per user), its facilities are oriented towards resource usage accounting. Thus, a system administrator can employ psacct to produce nice reports about the number of CPU hours spent per command or user. Figure 5.3 shows sample psacct output from the Linux operating system.

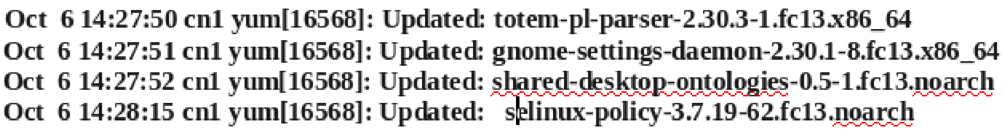
Syslogd [75] and Winsyslogd [76] are examples of widely employed Security Event Manager (SEM) applications. An SEM aggregates various types of audit records into a single interface. Audit record aggregation means that information is accepted not only from operating system audit trails, but also from third party sources such as security tools or even software application logs. The interface could be as simple as a human readable text file (Figure 5.4) or it can have its own sophisticated Graphical User Interface (GUI), as shown in the screenshot of Figure 5.5.



```
#Displays resource statistics per executed shell command
    #    * sum of system and user time in cpu second
          #   * "real time" in wall clock minutes
    #    * sum of system and user time in cpu minutes
     #    * cpu-time averaged core usage, in 1k units
                     #   * command name
          1034  17545.52re     0.25cp    7754k
           16       0.25re           0.21cp   68656k  yum-updatesd-he
           23   5055.87re     0.01cp    6667k   ***other*
            8   3099.25re     0.01cp   16921k   sshd*

                        # sa -m
       #Displays resource statictics per system user
root                    987   8263.81re     0.23cp    7467k
toma                     45   9289.05re     0.01cp   11965k
 nickb                   18      0.03re     0.00cp   15088k
```

Figure 5.3: psacct based resource auditing

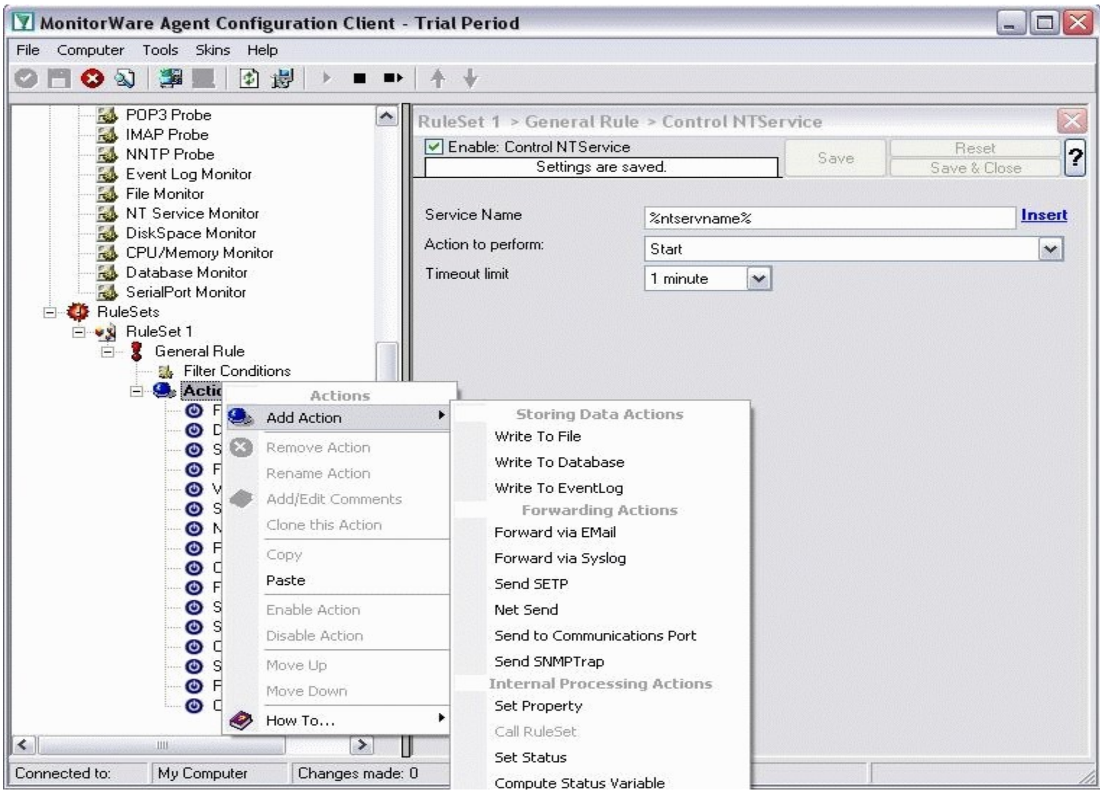


```
Oct  6 14:27:50 cn1 yum[16568]: Updated: totem-pl-parser-2.30.3-1.fc13.x86_64
Oct  6 14:27:51 cn1 yum[16568]: Updated: gnome-settings-daemon-2.30.1-8.fc13.x86_64
Oct  6 14:27:52 cn1 yum[16568]: Updated: shared-desktop-ontologies-0.5-1.fc13.noarch
Oct  6 14:28:15 cn1 yum[16568]: Updated:   selinux-policy-3.7.19-62.fc13.noarch
```

Figure 5.4: Syslog based audit record aggregation on a plain text file

This is a small sample of audit record engines and SEM tools. One obvious thing to observe is that there is not a consistent audit record format amongst these log engines. This format diversity might suit specific operating system environments but it creates many problems, especially when one needs to devise a mechanism to consolidate logs from different operating systems and resources.

Bishop [77] was one of the first to discuss these issues in the context of distributed audit record engines and to propose various solutions for standardized audit record formats.



RuleSet 1 > General Rule > Control NTService

☑ Enable: Control NTService

Settings are saved.

Service Name: %ntservname%

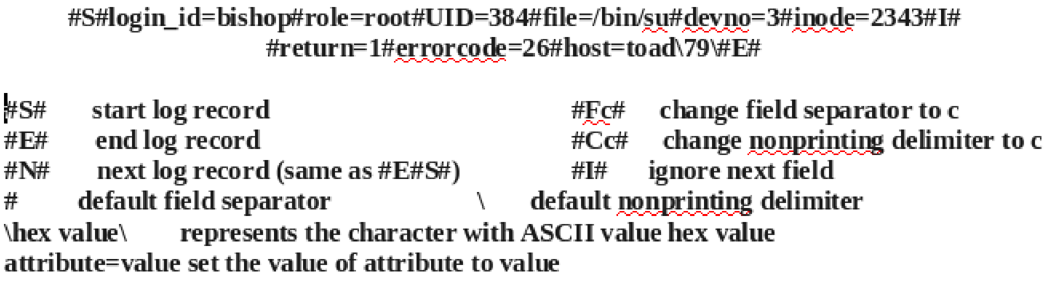
Action to perform: Start

Timeout limit: 1 minute

Figure 5.5: WinSyslog event management console

Figure 5.6 shows a sample of a proposed standard audit record format by Bishop [77], together with special purpose plain text ASCII based [78] field separators. The purpose of these field separators is to maintain compatibility amongst the character encoding sets of different operating systems. Whilst many characters encoding differences have now been addressed by the Unicode standard [79], Bishop's work is an interesting reminder of the need for a standard audit record format.

Another notable research effort in the field of producing suitable audit record engines is that of Larson et al [80]. The authors present a tool that automatically extracts attack manifestations from audit records, as shown in Figure 5.7. The process of comparing data generated during normal operation to data generated during a successful attack enables METAL to identify all processes that may be affected by the attack and the specific system call sequences, arguments and return values that are changed by the attack. Hence, it is possible to analyze many attacks in a reasonable amount of time, as well as locating groups of attacks with similar properties. Their approach proves how important is the design and format of audit records for detecting (and thus describing) insider threats. It also shows that it is possible to calibrate the content and format of audit records, by looking at particular threat types.



#S#login_id=bishop#role=root#UID=384#file=/bin/su#devno=3#inode=2343#I#
#return=1#errorcode=26#host=toad\79\#E#

#S# start log record
#E# end log record
#N# next log record (same as #E#S#)
default field separator
#Fc# change field separator to c
#Cc# change nonprinting delimiter to c
#I# ignore next field
**** default nonprinting delimiter
\hex value represents the character with ASCII value hex value
attribute=value set the value of attribute to value

Figure 5.6: Standard audit record format by Bishop

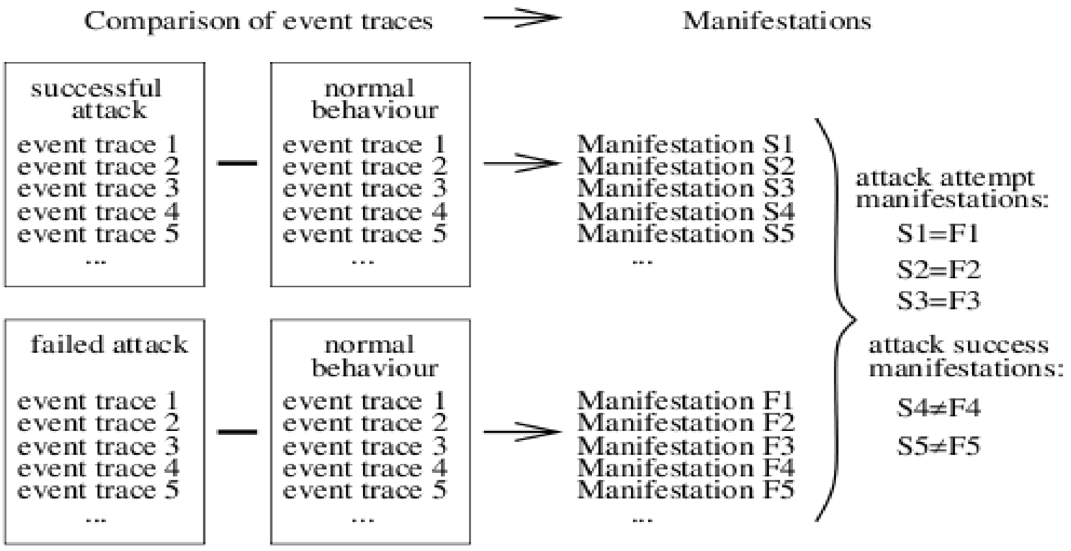


Figure 5.7: Manifestation Extraction Tool for Analysis of Logs (METAL)

Looking back at the previously discussed audit system approaches, serious deficiencies can be located in terms of using them for insider threat prediction. These deficiencies can be inferred by means of assessing the existing audit record engine features with the ITPSL functional requirements of Section 4.3 of the thesis and some of the CC requirements discussed in the earlier paragraphs of the current section.

Firstly, we have issues that concern the bridging of the format variability (structure and content) that break ITPSL requirements FR6 and FR7. Modern SEMs might consolidate information from various different devices and operating system vendors, but they are far from describing sufficiently issues in an operating system agnostic way.

In addition, process accounting tools might not cover sufficiently the variety of different system level information (file, process execution and network level), invalidating ITPSL requirement FR3. In fact, some of them might miss data as described in [81].

Several audit record systems do not report consistently the timing of audit record generation. For instance, many implementations of the syslog audit standard [75] and psacct tools [74] generate the audit record by entering the time stamp of the client system. If the client system does not have a reliable time source, this generates inaccurate information and thus conflicts with the temporal indicator aspect of ITPSL requirement FR4.

Moreover, some audit record engines might not meet the scalability and data integrity requirements set by CC requirement FAU_STG.1. Syslog [75] will not always consolidate data in a central location away from the audited client. The same can be said for the BSM standard [72] [73], leaving the integrity of audit data at risk. In addition, storing data in binary or text files might raise issues of storage efficiency and scalability.

Finally, one of the most serious drawbacks of existing audit approaches is the inability to store the audit information in form that can perform relational queries. This is necessary to satisfy ITPSL requirements FR4 and FR5 that concern the core functionality of the language semantics.

For all these reasons, this research project has designed and built a prototype audit record engine from scratch. This prototype should address most of the previously discussed issues and provide the data layer that the ITPSL can query. It should also act as a complement for existing forensic tools and provide the researchers enough information to replay/reconstruct insider threat scenarios. The next and final chapter section will present the design and features of this special audit engine.

5.3 The LUARM audit engine

This chapter started by explaining the vital relation between the audit record and the insider threat specification/prediction process. The previous section presented existing audit engine standards and found then unsuitable to meet all the proposed ITPSL requirements, as discussed in Section 4.3.

Beyond addressing the ITPSL requirements of Section 4.3 and the discussion of existing audit log engine deficiencies of the previous section, the fundamental problem that an insider misuse audit log engine has to tackle is how to reliably correlate data from different audit levels. Figure 5.8 displays an abstract view of a computer system audit log, in order to demonstrate the need to clarify how information from different audit data levels can be correlated.

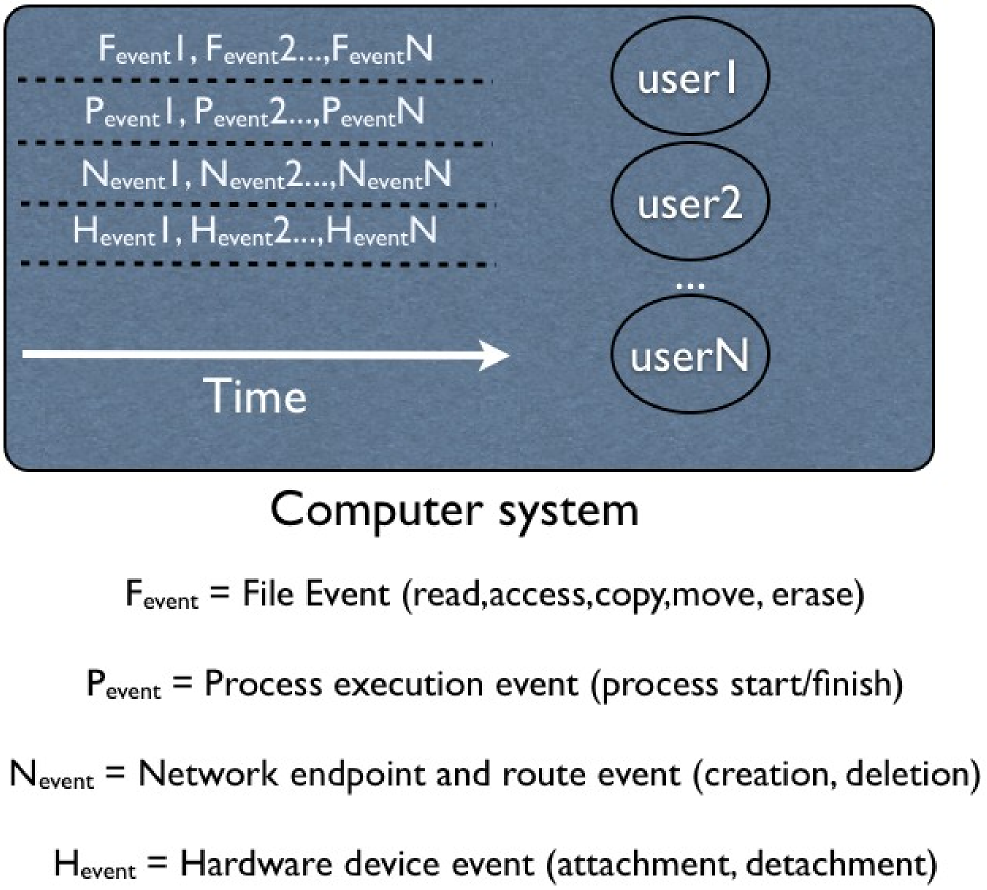

Fevent1, Fevent2...,FeventN
Pevent1, Pevent2...,PeventN
Nevent1, Nevent2...,NeventN
Hevent1, Hevent2...,HeventN
user1
user2
...
userN
Time
Computer system
Fevent = File Event (read,access,copy,move, erase)
Pevent = Process execution event (process start/finish)
Nevent = Network endpoint and route event (creation, deletion)
Hevent = Hardware device event (attachment, detachment)

Figure 5.8: An abstract view of a computer system audit log

A number of time ordered discrete events at the filesystem (F_{event}), process execution (P_{event}), network (N_{event}) and hardware monitoring level (H_{event}) can be used to shape a picture of what is happening in a computer system. The Magklaras and Furnell taxonomy [37] justified the role of the different audit levels. The separation of audit levels is also useful because each level requires different mechanisms to reliably record information. Later paragraphs are going to discuss these mechanisms in detail.

Apart from the time series of the four audit event types, Figure 5.8 displays the discrete user entities (User1...UserN). Their role is important in the abstraction schema because they must be relate to discrete audit events, in order to aid the task of accountability. In plain words, the audit record structure needs to ascertain statements such as:

- Network endpoint event Neven3 was started by process event Pevent7
- User entity User3 accessed file X according to Fevent20, which resulted in the launch of program Y (Pevent135) which created two network connections at port TCP 80 and 443 (Nevent 200,Nevent201).

Thus, it is important not only to interrelate different types of events, but also to link with certainty user entities to various related events. None of the currently existing audit/log engines is built to provide that level of event and user entity correlation.

The proposed engine addressing the previously mentioned deficiencies is called LUARM, which stands for Logging User Actions in Relational Mode [82]. It is an Open Source audit record engine that uses a Relational Database Management System (RDBMS) [83] for the storage and organization of audit record data.

The employment of an RDBMS system is a core design choice for the LUARM engine. It offers the necessary data availability, integrity and scalability features, as discussed in previous paragraphs, because most RDBMS tools are explicitly designed to organize and store large amounts of data [83]. However, the main reason of placing an RDBMS engine at the core of LUARM is the ability to have a tremendous flexibility in the process of querying audit records in a standard manner. The Structured Query Language (SQL) [84] is a declarative computer language used to query and process the data stored by RDBMS systems. SQL adheres to the relational model [66] and this means that it can easily match ITPSL requirements FR2 (signature repository creation) and FR4.

In terms of ITPSL FR4, the reader can recall Meier's feature specification list for misuse detection languages [36] (Section 3.3). Most of the event pattern dimension expression requirements can be facilitated by the SQL standard. In particular, features such as the disjunction, conjunction and negation operators are part of the language. SQL calls these predicates and it used them to specify conditions in an accurate manner. Boolean (true/false/unknown) truth values are used to limit the effects of statements and queries. In addition, step instance selection and completion, as well as data correlation can be performed by using SQL clauses such as 'FROM' and 'WHERE'. Later paragraphs provide LUARM examples using standard SQL queries.

Figure 5.9 displays the module client-server architecture of the LUARM audit engine. On the left of the figure, we can see a set of audited computer clients. Every client is running a unique instance of a set of monitoring scripts. Each of the client scripts audits a particular system level aspect of the operating system: 'netactivity.pl' audits the addition and creation of endpoints, 'fileactivity.pl' records various file operations, 'psactivity' provides process execution audit records and 'hwactivity' keeps a log of hardware devices that are connected or disconnected from the system. The right hand side contains the centralized server part of the architecture where audit data are stored, maintained and queried in a MySQL [88] based RDBMS. The Perl programming language [89] is used to implement the modules and the communication between client and server is performed via a Perl DBI [90] interface. Appendix A contains the table schema structure and the code for LUARM.

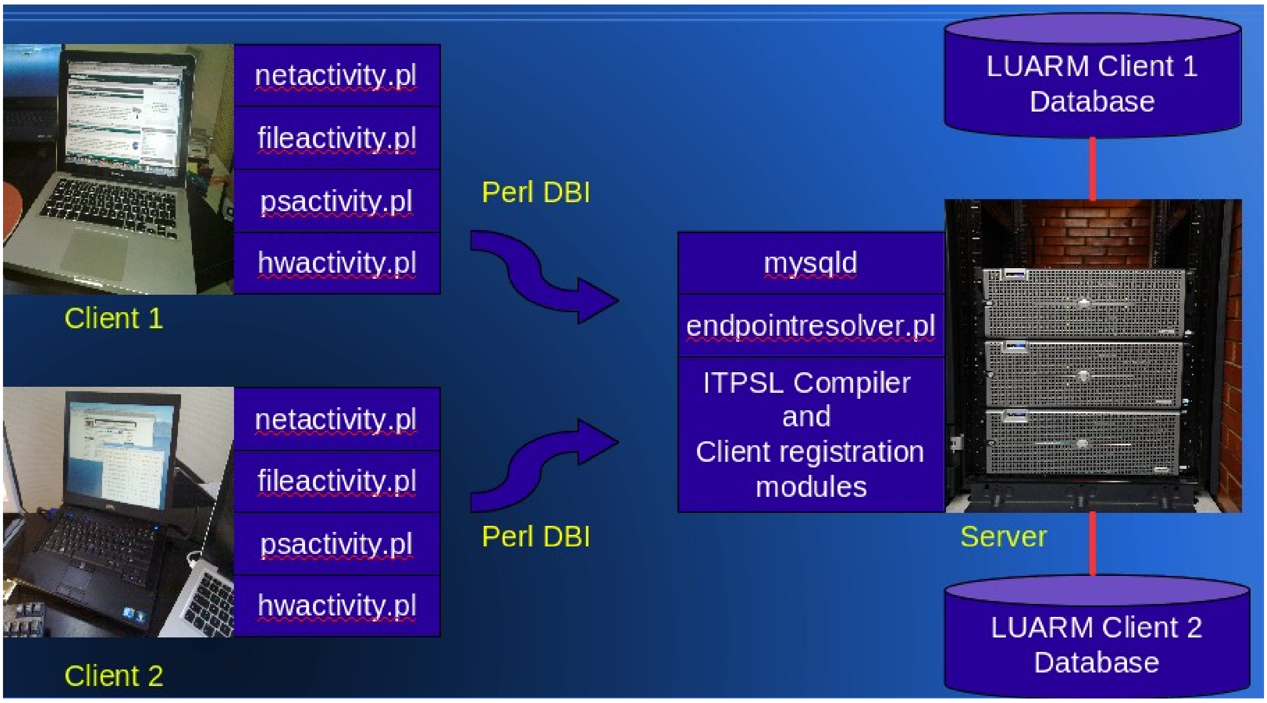


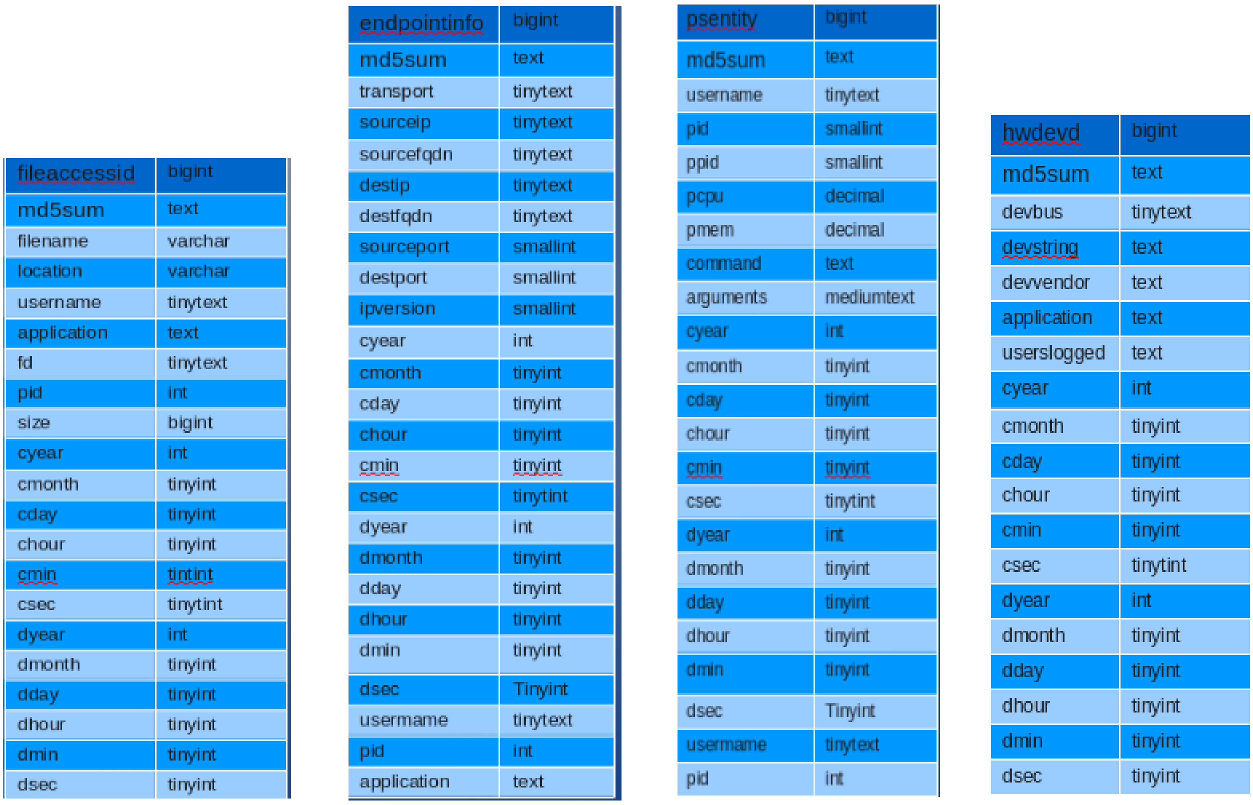
Figure 5.9: The LUARM architecture

The client-server architecture avoids leaving the data in vulnerable clients. To prevent issues that affect the scalability of operations and satisfy data access control isolation (addressing the CC [70] requirements FAU_SAR.1 and FAU_STG.1). The central host MySQL server has its own authentication system responsible for controlling who has access to the audit data. By authenticating audit reviewers against the RDBMS authentication system, we de-couple the users being audited from the auditors, a desirable property that ensures that audited insiders cannot easily manipulate audit data. Furthermore, by assigning a separate database instance per audited client, we reduce the likelihood of compromising the data for all clients (if the database access credentials of one client are compromised, the damage is limited to the audit data for that client only).

The client scripts were designed to examine every aspect of system level events, as dictated by the Magklaras and Furnell insider misuse taxonomy in [37]. Figure 5.10 displays the relational table format for the four types of recorded audit data in LUARM: fileaccess, process execution, network endpoint and hardware device information.

Temporal information is provided by event creation time stamps (cyear, cmonth, cday,chour,cmin,csec) and respective event destruction time stamps (dyear, dmonth,dday,dhour,dmin,dsec). The combination of the two types of timestamps can pinpoint exact time intervals for events in a consistent format for all recorded event types. In contrast, most audit systems may provide only event creation time references without hinting for the duration of an event.

The sampling frequency for recording events is that of a 100 milliseconds. Although the sampling frequency can be adjusted in the various LUARM modules, a value of 100 milliseconds was an intentional decision. At first, this might seem problematic as many attack steps can occur in sub second times. However, time resolution varies amongst operating systems. In Linux, the finest granularity of timing for most computing devices varies from approximately 10 milliseconds all the way down to a 1 microsecond, depending on the hardware details [85]. The Windows 7 operating system (and its various derivatives) has a timer granularity of 15.6 ms [86]. For these reasons, LUARM relies on the Perl Time::HiRes module [87] to bridge the gap between the different operating system timer implementations. A 100 millisecond sampling frequency is also a good compromise between accuracy and scalability. The more granular the time resolution, the greater the computational load for both the client and the server LUARM parts.



fileaccessid	bigint
md5sum	text
filename	varchar
location	varchar
username	tinytext
application	text
fd	tinytext
pid	int
size	bigint
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tintint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	tinyint

Fileinfo table

endpointinfo	bigint
md5sum	text
transport	tinytext
sourceip	tinytext
sourcefqdn	tinytext
destip	tinytext
destfqdn	tinytext
sourceport	smallint
destport	smallint
ipversion	smallint
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	Tinyint
usermame	tinytext
pid	int
application	text

Netinfo table

psentity	bigint
md5sum	text
username	tinytext
pid	smallint
ppid	smallint
pcpu	decimal
pmem	decimal
command	text
arguments	mediumtext
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	Tinyint
usermame	tinytext
pid	int

Procinfo table

hwdevd	bigint
md5sum	text
devbus	tinytext
devstring	text
devvendor	text
application	text
userslogged	text
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	tinyint

Hwinfo table

Figure 5.10: LUARM audit data table format

Each audit record of an event table is identified by a unique table key of bigint MySQL type [88]. In version 5.1 of the MySQL RDBMS, a 'bigint' numeric type can create up to 18446744073709551615 unique keys, a number large enough to archive a useful number of events in each LUARM event table.

Another important design decision that concerns the format of the audit table was to include common attributes amongst different event tables for the purposes of increasing the ability to correlate events and provide user entity accountability. This is in response to ITPSL requirement FR8 and CC [70] requirements FAU_GEN1.2 and FAU_SEL1.1. For instance, fields such as 'username' (user entity), pid (numeric process ID of the program responsible for the event creation) and application (string that represents the name of the application that matches the pid) can be found in most of the event tables. This enables the audit reviewer to use SQL and relate events, so he can form queries of the type “Find the network endpoint created by program x of user y” in an easy manner.

The use of the MD5 cryptographic hash function [91] (md5sum field) is used on all event tables for performing audit record updates in an efficient manner. In particular, every time LUARM inserts an audit record in a table, it calculates an MD5 sum of several relevant table fields, in order to uniquely identify the event and keep track of the record being inserted in the database. On the next audit record insertion cycle, LUARM generates an MD5 sum of the live records and compares them to the stored MD5 sums of every active stored record (a record that has a NULL value for the d* timestamps). If the MD5 sums do not match the record is inferred as a new one and is inserted to the database. This is a more efficient way than comparing multiple fields, in order to perform record updates. For more information about the implementation of record updates, the reader should consult Appendix A.

It should be noted that MD5 hash functions do exhibit cryptographic vulnerabilities, as their collision resistance is limited [92] [93]. This means that as the size of the MD5 function input increases, so does the probability of having the same MD5 sum output for different inputs (collision). This could pose a problem in the LUARM mechanism for record updates. . The collision resistance of the MD5 hash function is estimated to $2^{20.96}$ [93]. In practical terms, this means that there is strong probability to find a pair of inputs that produce the same hash value roughly for every 2039805 inputs. This would erroneously disregard a record and thus miss important system activity information. However, the size, as well as the number of records stored for every monitored client is in the order of hundreds of thousands for a period of several months. For this reason, we conclude it is safe to rely on MD5 hashes for the moment and consider shifting to another more collision resistant hash function in future LUARM versions.

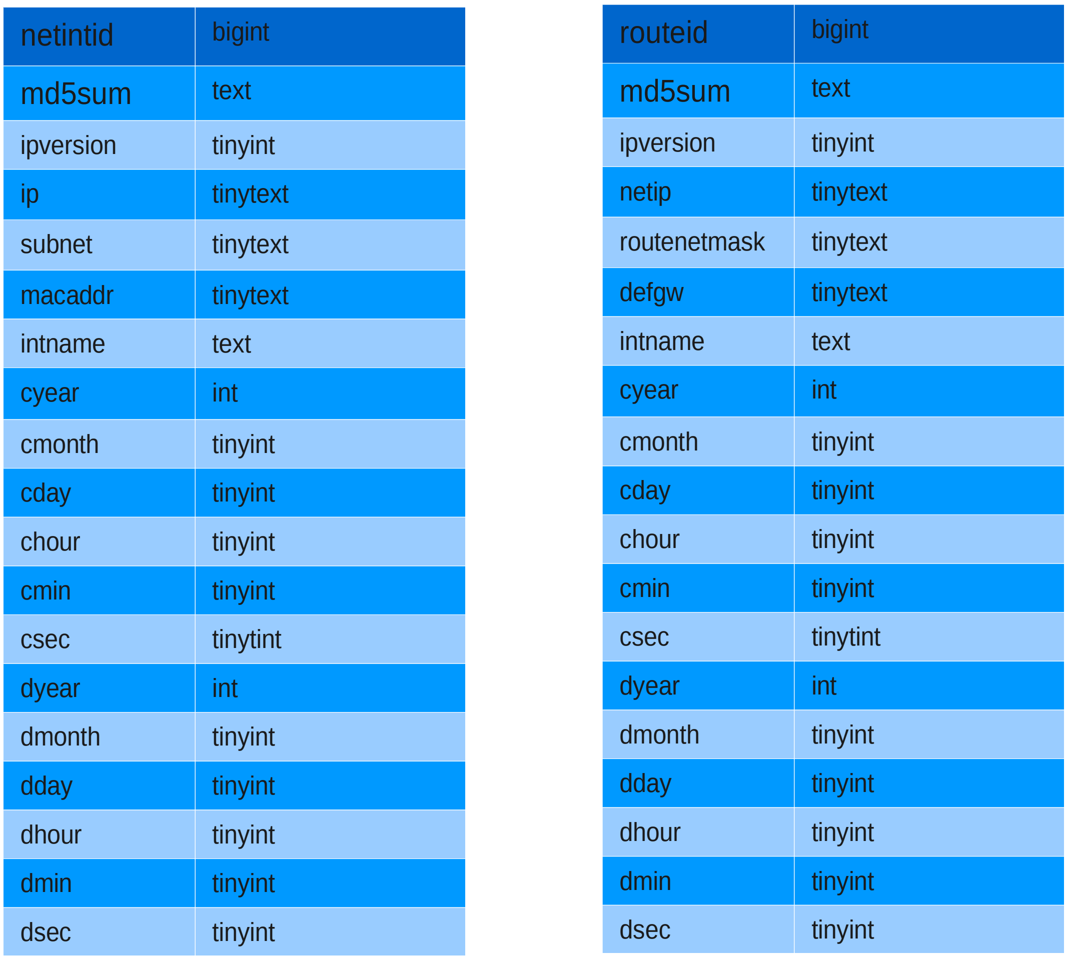
The Magklaras and Furnell taxonomy [37] specified that file operations are an important part of misuse specification. The 'fileinfo' table stores file access related events. The filename specification consists of two parts. The 'filename' field which holds the filename with the file extension (i.e. data.txt) and the 'location' field which contains the absolute path of the file. The fact that the two are divided in separate fields makes it easier to search by location or by field name only, increasing the versatility of mining file data. In order to populate the data on this table, LUARM relies on the 'lsof' utility [94]. The utility is versatile and can record a variety of events including file and network endpoints in real time. It exists for an entire range of UNIX/Linux and Mac OS X operating systems, covering a large spectrum of computing devices. The Windows operating system family has 'psFile', a utility that it is part of the PsTools package [95] and offers the equivalent functionality of 'lsof'.

The 'netinfo' table covers the network operations part of the proposed taxonomy [37]. It logs the creation and destruction of network endpoints. In the context of LUARM, the term 'network endpoint' refers to the operating system data structures employed to facilitate network connectivity via the TCP/IP protocol suite [96]. Network endpoint activity is considered as live forensic data. A series of table fields are used to record endpoint details ('sourceip', 'destip', 'sourceport' , 'destport' and 'transport' record source and destination IP addresses, source and destination port and transport protocol respectively). The fields 'sourcefqdn' and 'destfqdn' hold the DNS [97] resolved Fully Qualified Domain Name (FQDN) for the source and destination hosts.

A small LUARM implementation detail concerning the 'sourcefqdn' and 'destfqdn' fields is that they are not populated by the client LUARM routines. In contrast, they are populated on the LUARM server side. Due to the criticality of correct DNS data for the audit records, the frequent DNS configuration errors [98], aspects of DNS operational security [99] and client performance, the endpoint name resolution is left on the server side. This provides a greater control on DNS derived data and does not rely on vulnerable clients (malicious insiders or software vulnerabilities) for auditing network connections.

Network endpoint auditing is not the only “live” network activity data recorded by LUARM. Figure 5.11 shows the structure of the 'netint' and 'netroute' tables. They record network interface and routing information respectively. The recording of network interface information allows the audit reviewer to know to which networks a particular device has been connected to. This is particularly useful for mobile devices (i.e. laptops), where it might be useful to know the location of a device. The network routing information can be used to correlate information of network endpoint activity, or the transmission of information via insecure or unauthorized networks, in connection with file and process execution activity.

Aspects of the Magklaras and Furnell insider misuse taxonomy [37] require the recording of network connection content or payload. For example, Figure 4.4 mentions the example of suspicious SMTP traffic in terms of email addresses and attachments. Email attachments and addresses are placed in the payload section of a TCP/IP packet. Inspecting the payload of network packets is the job of Network based Intrusion Detection Systems [25]. However, Schneier [100] doubts about the feasibility of network packet payload inspection. In an era of increasing network speeds and Virtual Private Network (VPN) technologies [101], decoding network payload in near real time to infer threats from the content of the payload can be a computationally expensive or impossible process. For these reasons, LUARM does not record payload content. It focuses instead on network endpoint presence and creation and the correlation of network data with file and process execution activities.



netintid	bigint
md5sum	text
ipversion	tinyint
ip	tinytext
subnet	tinytext
macaddr	tinytext
intname	text
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	tinyint

routeid	bigint
md5sum	text
ipversion	tinyint
netip	tinytext
routenetmask	tinytext
defgw	tinytext
intname	text
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	tinyint

Figure 5.11: Recording network interface and routing information

Process execution activity is recorded in the 'Procinfo' table (Figure 5.10). This table records 'live' forensic data. The table includes both the proces ID ('pid') and parent process id ('ppid'), so that process execution flow can be traced back to the original process. In order to speed up process execution searches, the LUARM engine also separates the executed command ('command') from its arguments ('arguments'). One might like to search them separately in the process of mining process execution data. The 'pcpu' and 'pmem' fields address process over-utilization issues, as described in Figure 4.3 ([37]). 'pcpu' contains the CPU time used divided by the time the process has been running (cputime/realtime ratio), expressed as a percentage. 'pmem' is the ratio of the process's resident set size to the physical memory on the machine, expressed as a percentage. The 'ps' UNIX/Linux utility [94] is used to collect process information. For all active processes (whose d* temporal fields are NULL), LUARM updates in near real time these two fields.

The 'hwinfo' table logs 'live' device connection and disconnection events, in response to the 'hardware' OS consequence of the Magklaras and Furnell taxonomy. All events generated by devices that connect to the Peripheral Component Interconnect (PCI and PCI-Express) and Universal Serial (USB) buses [102]. These two buses are commonly found on a large array of computing devices, interconnecting various peripherals such as portable storage media, as well as sound and video interfaces amongst others. For instance, an audit reviewer or forensics analyst might be interested to correlate file activity to a portable storage medium connection, as part of an intellectual property theft scenario. In that case, the 'hwinfo' table logs information in various fields that help identify the attached device ('devstring', 'devvendor'), the bus the device was connected to ('bus') and correlate the device attachment event against a number of users that are logged into the system at the time of the device attachment ('userslogged').

An additional number of LUARM tables help the engine perform housekeeping functions, such as keeping details of the registered host, the number of registered users. As these are not core functionality issues, the reader can refer to Appendix A for more details.

Having a proposed structure and content for the various categories of audit events, we can now issue sample SQL statements to illustrate how audit data mining is performed. Figure 5.12 displays sample queries that demonstrate the expressiveness of LUARM's audit record content and structure.

Find all accesses of the file 'prototype.ppt' by users 'toms' OR 'georgem' between 9:00 and 14:00 hours on 23/10/2009.

```
SELECT * FROM fileinfo WHERE filename='prototype.ppt' AND ((username='toms') OR (username='georgem')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour <= '13' AND cmin >= '0' AND cmin >= '59';
```

Find all USB devices that were physically connected to the system when users 'toms' OR 'georgem' were logged on 23/10/2009.

```
SELECT * from hwinfo WHERE devbus='usb' AND ((userslogged RLIKE 'toms') OR (userslogged RLIKE 'georgem')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour <= '13' AND cmin >= '0' AND cmin >= '59';
```

Find whether users 'georgem' or 'toma' have tried to move or copy a file called 'prototype.ppt' (irrespective of location) under the directory '/media' between 9:00 and 13:00 hours on the 23rd of October 2009.

```
select * FROM psinfo WHERE ((command='cp') OR (command='mv')) AND (arguments RLIKE 'prototype.ppt' AND arguments RLIKE '/media') AND ((username='georgem') OR (username='toms')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour <= '13' AND cmin >= '0' AND cmin >= '59';
```

Figure 5.12: Using SQL to mine data in LUARM

There are a few important observations to make about the example LUARM SQL queries. The first one concerns the embedding of system specific knowledge inside the statement. For instance, the third SQL statement example of Figure 5.12 defines in essence a step of an insider trying to transfer a sensitive file to a portable medium. One has to know the name of the sensitive file 'prototype.ppt' and also the fact that '/media' is used as a mount point for portable media for that host. Additional possible destination locations could be specified by means of OR operators. This is a nice example of how the structure of an SQL relational query fits Meier's event pattern expression requirements [36].

An additional point relates to the use of the 'RLIKE' operator [103], always in relation to the second and third examples of Figure 5.12. The operator implements a regular expression type of match [104]. Apart from the conjunction operator (OR), regular expressions give the specification polymorphic properties (one specification string, many matching results), a desirable property for compact misuse detection language statements. In Chapter 6 we will discuss the proposed ITPSL semantics and elaborate more on the issue.

Finally, the last observation is despite the powerful abilities of the SQL interface, writing SQL statements is a quite cumbersome process. A more user friendly interface is required that will be more expressive and will allow for easy event correlation. This is the territory of the ITPSL semantics.

5.4 Conclusions

In summary, this chapter has presented LUARM, a logging engine with an audit record specially designed to store insider misuse system-level oriented information. LUARM addresses a number of deficiencies found in commonly employed logging mechanisms. These deficiencies make the existing logging engines unsuitable for the task of insider IT misuse auditing. LUARM addresses those deficiencies and it follows the derived IT misuse taxonomy (Chapter 4) to shape an audit record format that provides user accountability. The next chapter discusses further details about the ITPSL construction process and presents the proposed semantics.

Chapter 6 The Insider Threat Prediction and Specification Language

The previous chapter addressed objective 3 of this research project (Section 1.2) by producing an audit engine tailored to the process of insider threat specification. The fourth thesis objective is the subject of this chapter. Having defined the problem domain, a suitable taxonomy and the structure and form of audit records, it is now time to introduce the Insider Threat Prediction and Specification Language (ITPSL), the core deliverable of the thesis.

The chapter will start by presenting ITPSL as a Domain Specific Language (DSL), an important programming concept employed as a guide to build the language. The second chapter section introduces the notion of semantics and discusses the methodologies to describe them as part of a language design process. The section concludes with the crucial choice of the translational semantic description methodology as a tool for the design of ITPSL semantics. The third chapter section follows with a justification of the reasons ITPSL was chosen to be an XML [122] based language. The section also provides an introduction of XML based technologies applicable to ITPSL. The rest of the chapter sections detail the ITPSL semantics in the context of describing insider misuse incidents and threats with examples that clarify their use in various simple scenarios.

6.1 ITPSL as a Domain Specific Language

The ITPSL scope defines clearly a specific task of expressing insider threat metrics. This paves the way for the selection of a mechanism that allows the language designer to focus on the problem in question. A Domain Specific Language (DSL) is a semantic mechanism tailored specifically for describing the details of a particular task. The main goal is the usage of appropriate semantics to reduce the effort required to reference and manipulate elements of that particular domain.

Spinellis [105] defines a Domain Specific Language as “programming language tailored specifically to an application domain: rather than being for a general purpose, it captures precisely the domain's semantics”. DSL schemata have been employed successfully in a number of different areas. Consel [106] discusses the range of applications that have employed a DSL which includes device driver construction, active networking and operating system process scheduling. Moreover, Eric Raymond [107] outlines some widely known ‘mini’ languages employed in the Unix community (regular expressions, awk, m4) and beyond (Postscript, SNG, Glade) as examples of domain specific languages. This list is by no means exhaustive, as many more DSLs exist today. A DSL is really a framework that offers the ability of building specific and concise notations to express a problem domain, as well as safe (as predictable) code due to semantic restrictions. Both of these properties are very desirable in the process of developing insider threat specifications.

DSLs are also categorized as external and internal in terms of the way they are implemented [108]. External DSLs are discrete systems, independent from any host language and they contain their own interpreter or compiler to parse the language statement and perform post interpretation/compilation actions. In contrast, internal DSLs are semantics embedded inside a general purpose programming language and thus are dependent from the interpreter/compiler of the host language. Examples of external DSLs are the ‘mini’ Unix languages mentioned by Raymond in [107], whereas internal DSL languages tend to be embedded in programming languages such as Lisp [109], Smalltalk [110] and Ruby [111]. Also notable is the new Grammar Engine (PGE) of 6th version of the Perl language [112] that allows one to construct localized version of the Perl language and could form the basis to build DSLs.

The process of deciding which DSL approach to follow for implementing ITPSL is important. External DSL approaches offer a greater freedom to experiment with the process of constructing insider threat semantics but they provide a higher overhead when it comes to development issues combined with a higher learning curve for the language users. On the other hand, internal DSLs offer less development overhead as parsing, interpretation and compilation issues are handled by the host language environment. If one takes into account that the host general programming language will have already mature semantics and an established user base, it is easy to conclude that an internal DSL would have less steep learning curve than an external DSL approach.

However, the internal DSL dependency on the host language environment might create problems for the language designer. The most important issue might arise from a mismatch between the symbolic integration of the embedded DSL and the general vocabulary of the general purpose host language. General purpose language vocabularies are rich enough to express a variety of scenarios in an abstract way. For example, on a network access scenario, a general purpose programming language vocabulary can express details of the origin and destination of a network connection but not express network access patterns. In that case, if one tries to engineer the additional functionality into the general language, the process of constructing meaningful semantics might be impaired due to the general language syntax or due to the host language underlying data structures that might not be able to represent fully the required domain information.

A secondary practical problem of adopting an internal DSL approach might include parameter evaluation and performance issues. An insider threat prediction operational environment requires the evaluation of various parameters at runtime. If a statically compiled host general language is used (such as C/C++), runtime evaluation of parameters might pose a challenge. There are of course scripting languages [113] where runtime evaluation is not an option, but they might be slow. Ways to combine compile and runtime languages do exist (i.e. a Perl Script calling a C/C++ library via API wrappers), however the complexity of combining domain specific semantics with more than one language should not be underestimated.

For all these reasons, ITPSL follows the external DSL approach allowing for freedom to create the semantics from scratch with commonly changed parameters to be altered without recompilation issues and no dependence on host language idiosyncrasies. The issue of the learning curve for a domain expert to learn yet another language is of course considerable. However, the narrow scope of a DSL language combined with carefully crafted semantics should create a low complexity interface of relatively few (when compared to a general purpose language) statements and thus make the language easy to learn. This approach has been followed by a number of security related research DSLs such as CISL [17] and Panoptis [18], as discussed in chapter 2. For now, it should be noted that both of them can be categorized as external DSLs using configuration files to encode statements that have no resemblance to general purpose programming languages.

There are also a number of external DSLs that utilize XML to convey information. The next section discusses the use of XML as a markup to construct DSLs and shows that this is a common approach.

6.2 Designing the ITPSL semantics

Prior describing the ITPSL semantics, it is important to explain the notion of the term 'semantics'. In broad terms, semantics is simply another word for meaning. However, the term encompasses many important concepts that merit a careful description.

Ogden and Richards, two important linguists and philosophers of the first quarter of the 20th century wrote a book entitled “The meaning of the meaning” [114]. Their work describes important ideas of the semantic and semiotic fields [115]. They state the idea of the “meaning triangle”, a triangular relationship amongst ideas, linguistic symbols like words and the real world. In essence, every person links a word to a real world item through a mental concept. This means that semantics have a subjective nature and this fact is fundamental for the design of languages based on semantics.

Software language engineers relate semantics to the term language by using the following definition: “A description of the semantics of a language L is a means to communicate a subjective understanding of the linguistic utterances of L to another person or persons.” [116].

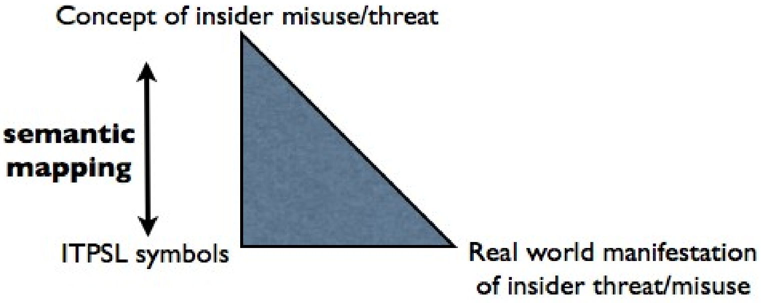

Concept of insider misuse/threat
semantic mapping
ITPSL symbols
Real world manifestation of insider threat/misuse

Figure 6.1: Semantic mapping in ITPSL and the meaning triangle

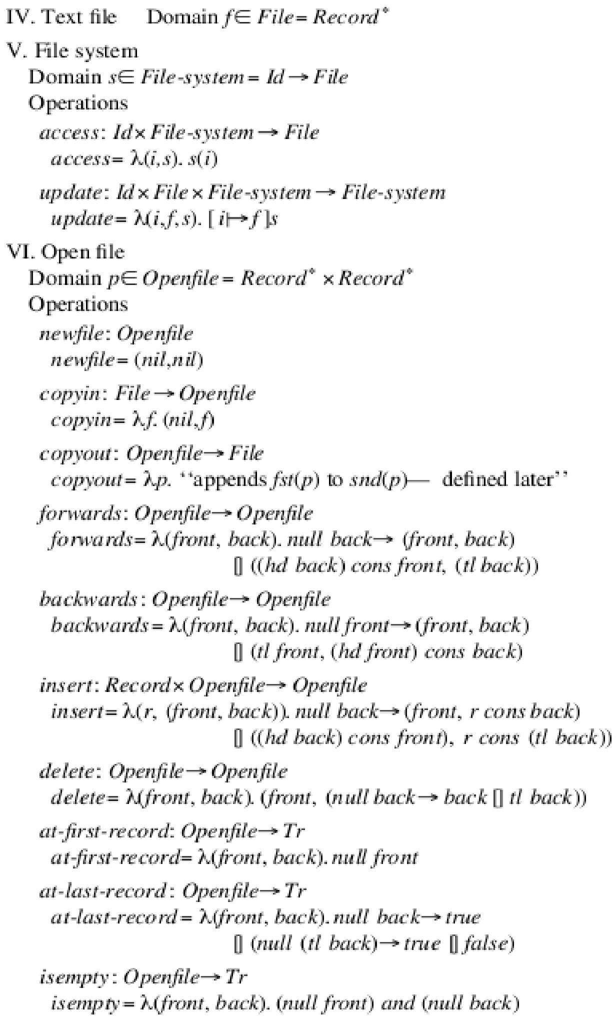
Whilst the previous definition makes sense for spoken (natural languages), the semantics of a software language (and thus a DSL) have a different scope. At present, computers do not have the capability of constructing their own mental reality. Hence, software language semantics target only humans, even if they describe aspects of computer system operations. For this reason, semantics facilitate the mapping between the human concept, the language symbols and what actually happens in the computer system.

Figure 6.1 illustrates the role of semantics in ITPSL. The semantic mapping is all about connecting the symbols of ITPSL to the human expert's view of how insider misuse incidents or threats occur in a computer system. The concept at the top is formed by both the insider threat taxonomy and models described in Chapter 4, as well as by LUARM's specialized audit record format discussed in Chapter 5. These represent human abstractions of the problem domain. On the right hand side of the triangle resides the real world system view. Thus, the hypotenuse of the triangle symbolizes the differences between the concept and the real world manifestations of threats and misuse incidents. Successful semantic mappings adapt to their target audience and reduce the distance between symbols and a well formed concept. This in turn results in a smaller hypotenuse indicating more accuracy for the concept with respect to what is actually happening in the computer system.

Having defined the role and scope of semantics in a problem domain, the next important question to raise is how one describes the semantics. This task is at the core of a language construction process. Software languages require formal methodologies to describe semantics, as it is important to eliminate ambiguities that occur in natural languages. The proposed insider threat model of Chapter 4, and the relational description of events facilitated by LUARM provide the formal foundation for the description of system-level events that can describe insider IT misuse and threats. However, the linking of the formal foundations to symbols is the next step. Choosing the right methodology at this stage is very important.

In general, there are four methods to formally describe semantics [116]:

- **The denotational method**: Mathematical objects (denotations) are constructed that represent the meaning of the chosen semantics.
- **The pragmatic method**: This involves the execution of the semantics by the 'reference implementation' tool, a program designed to produce specific outputs given specific semantic inputs.
- **The translational method**: This method translates the semantics into another language that is well understood.
- **The operational method**: A set of syntactically validated semantics is intepreted as a series of computational steps.



IV. Text file Domain $f \in \mathit{File} = \mathit{Record}^*$

V. File system

Domain $s \in \mathit{File\text{-}system} = \mathit{Id} \to \mathit{File}$

Operations

$\mathit{access}: \mathit{Id} \times \mathit{File\text{-}system} \to \mathit{File}$

$\mathit{access} = \lambda(i,s).\ s(i)$

$\mathit{update}: \mathit{Id} \times \mathit{File} \times \mathit{File\text{-}system} \to \mathit{File\text{-}system}$

$\mathit{update} = \lambda(i,f,s).\ [\,i \mapsto f\,]s$

VI. Open file

Domain $p \in \mathit{Openfile} = \mathit{Record}^* \times \mathit{Record}^*$

Operations

$\mathit{newfile}: \mathit{Openfile}$

$\mathit{newfile} = (\mathit{nil}, \mathit{nil})$

$\mathit{copyin}: \mathit{File} \to \mathit{Openfile}$

$\mathit{copyin} = \lambda f.\ (\mathit{nil}, f)$

$\mathit{copyout}: \mathit{Openfile} \to \mathit{File}$

$\mathit{copyout} = \lambda p.$ ''appends $\mathit{fst}(p)$ to $\mathit{snd}(p)$— defined later''

$\mathit{forwards}: \mathit{Openfile} \to \mathit{Openfile}$

$\mathit{forwards} = \lambda(\mathit{front}, \mathit{back}).\ \mathit{null}\ \mathit{back} \to (\mathit{front}, \mathit{back})$
$[]\ ((\mathit{hd}\ \mathit{back})\ \mathit{cons}\ \mathit{front},\ (\mathit{tl}\ \mathit{back}))$

$\mathit{backwards}: \mathit{Openfile} \to \mathit{Openfile}$

$\mathit{backwards} = \lambda(\mathit{front}, \mathit{back}).\ \mathit{null}\ \mathit{front} \to (\mathit{front}, \mathit{back})$
$[]\ (\mathit{tl}\ \mathit{front},\ (\mathit{hd}\ \mathit{front})\ \mathit{cons}\ \mathit{back})$

$\mathit{insert}: \mathit{Record} \times \mathit{Openfile} \to \mathit{Openfile}$

$\mathit{insert} = \lambda(r,\ (\mathit{front}, \mathit{back})).\ \mathit{null}\ \mathit{back} \to (\mathit{front},\ r\ \mathit{cons}\ \mathit{back})$
$[]\ ((\mathit{hd}\ \mathit{back})\ \mathit{cons}\ \mathit{front}),\ r\ \mathit{cons}\ (\mathit{tl}\ \mathit{back}))$

$\mathit{delete}: \mathit{Openfile} \to \mathit{Openfile}$

$\mathit{delete} = \lambda(\mathit{front}, \mathit{back}).\ (\mathit{front},\ (\mathit{null}\ \mathit{back} \to \mathit{back}\ []\ \mathit{tl}\ \mathit{back}))$

$\mathit{at\text{-}first\text{-}record}: \mathit{Openfile} \to \mathit{Tr}$

$\mathit{at\text{-}first\text{-}record} = \lambda(\mathit{front}, \mathit{back}).\ \mathit{null}\ \mathit{front}$

$\mathit{at\text{-}last\text{-}record}: \mathit{Openfile} \to \mathit{Tr}$

$\mathit{at\text{-}last\text{-}record} = \lambda(\mathit{front}, \mathit{back}).\ \mathit{null}\ \mathit{back} \to \mathit{true}$
$[]\ (\mathit{null}\ (\mathit{tl}\ \mathit{back}) \to \mathit{true}\ []\ \mathit{false})$

$\mathit{isempty}: \mathit{Openfile} \to \mathit{Tr}$

$\mathit{isempty} = \lambda(\mathit{front}, \mathit{back}).\ (\mathit{null}\ \mathit{front})\ \mathit{and}\ (\mathit{null}\ \mathit{back})$

Figure 6.2: Denotational semantics to describe an interactive file editor [117]

The denotational method [117] is based on complex mathematical constructs that resemble ideas of Object Oriented [118] and Functional [119] Programming paradigms. There are actors, objects to act on and domains, connected together to form concepts of computer system events by means of set theory operations. The method combines mathematical formalism and notational elegance in order to attack successfully the issue of natural language ambiguity.

Figure 6.2 provides an example of denotational semantics for the purpose of describing an interactive file editor. The overall concept is decomposed down to a series of operations on objects (file, filesystem) and each set of operations has its own domain. Despite the rigor of this method to describe systems effectively, it is fair to say that it is complex, as the intended audience is normally limited to mathematicians or computer scientists that have mastered the use of denotational semantics. The intended ITPSL audience consists of IT system professionals (system administrators and security specialists). It is not safe to assume that every IT system professional is (or will become) familiar with denotational semantics. Thus, using the denotational method as the basis for ITPSL semantics is a poor choice.

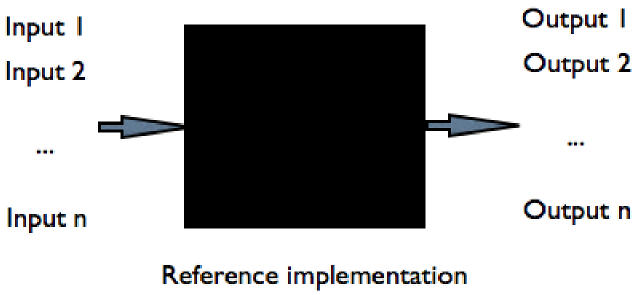


Figure 6.3: Pragmatic semantic description

Pragmatic semantic description is the simplest of the four previously mentioned semantic description methods. It can be best described by the illustration example of Figure 6.3. The black box is the “reference implementation” module that acts on the semantics. It takes various sets of discrete symbols as its input on the left and outputs pre-defined symbols on the right. The contents of this black box are not known. As a result, the only thing one can do to figure out the meaning of the input symbols is to execute various input combinations and observe the respective outputs. This method is normally used in the process of reverse engineering software systems [120].

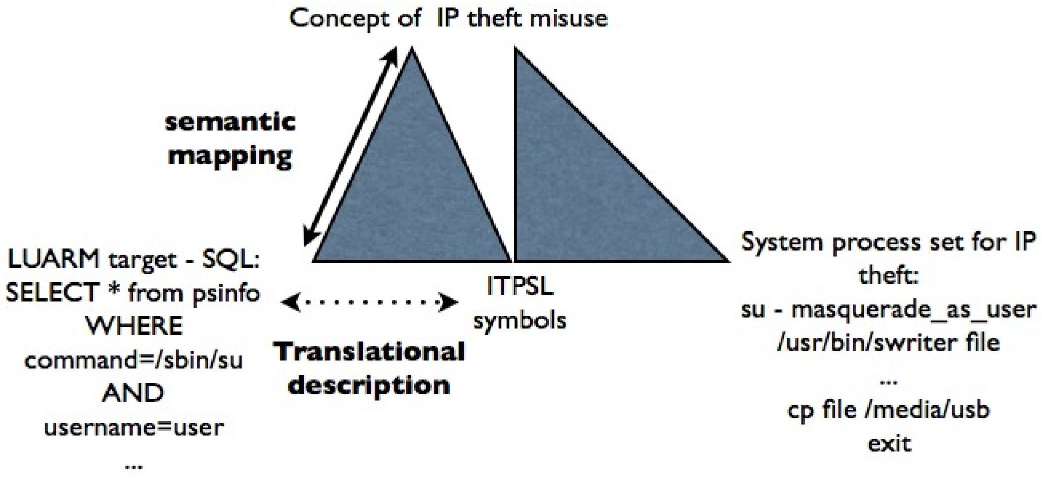


Figure 6.4: Translational semantic description in ITPSL

The downside of the pragmatic semantic description is that it requires a reference implementation and it lacks descriptive power. For example, it is not known how many inputs have to be issued, in order to get a sufficient amount of output to infer what actually happens inside the black box. The equivalent paradigm transferred to the ITPSL domain means that one would have to submit the ITPSL semantics to a reference implementation of the language compiler and then watch whether the compiler reacts on certain events. This might be desirable as a language validation mechanism. However, this is the reference implementation, the deliverable for the project and not the initial condition. Hence, the pragmatic semantic description is written off the list of desirable methodologies.

This leaves the options of translational and operational semantic description methodologies to choose from. There are many similarities between the two. The most important of them is that they both converge around the problem of translating a set of symbols into a known target language to facilitate semantic mapping. Figure 6.4 exemplifies the principle of these two semantic description mechanisms by using the mapping triangle concept of Ogden and Richards [114]. It displays what happens when one tries to specify an Intellectual Property (IP) theft scenario. In this scenario, an insider masquerades as another user, reads the sensitive file and then copies the file into a portable storage medium.

The isosceles triangle on the left represents the role of the translation procedure from the ITPSL semantics to the well known target language. The term 'well known' does not only mean familiar in this case. A target language is well known also because it can sufficiently represent the concept of the misuse/threat scenario in system related terms, bringing it close to the set of processes that represent the actual IP theft occurrence in the computer system (right hand side of the orthogonal triangle). For ITPSL, that target language is no other than the LUARM SQL schema [82], as described in Chapter 5. The LUARM audit record and the ITPSL symbols are implicitly known by the target audience because they can both represent the two parts of the run-time environment of the computer system: the data and the processes. It is this translation that maps a set of ITPSL symbols to a set of LUARM SQL statements which in turn try to relate to the real system.

Operational semantics are used in a similar manner. They do need a translation between language symbols and a well known target language. However, an operational semantic approach introduces the concept of state transitions [26]. State transitions can create a particular problem in the process of expressing computer system run-time environments: They separate the data and the processes. This can have profound implications for a specification language, as it forces the language user to describe both the run-time state and tie it to specific symbols. When one does not know the possible combination of states (aka processes that relate to a threat or misuse) and ITPSL symbols, it becomes virtually impossible to correlate a set of events and provide functions like threat prediction (not only misuse detection).

In formal computer science terms, ITPSL semantics need referential transparency, an important principle of functional programming [119] which allows an expression to replace its value without changing the program it is part of (in other words, yielding a program that has the same effects and output on the same input). This cannot be facilitated by an operational semantic description mapping approach. As a result, the only suitable methodology for describing ITPSL semantics is that of translational semantics.

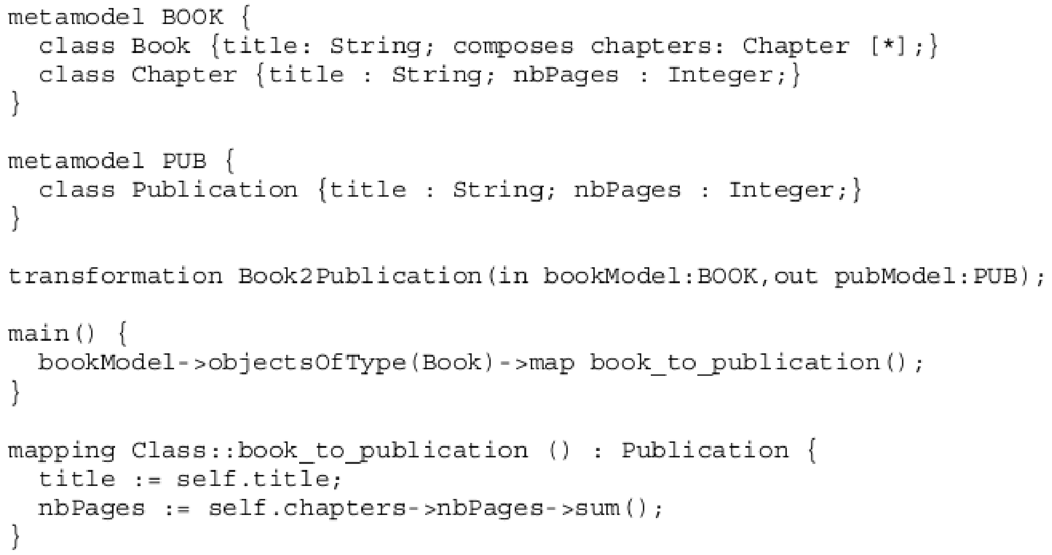
The next section discusses the thinking behind the choice of a suitable ITPSL translational semantic approach.

6.3 XML as the basis for ITPSL code generation

The previous section presented the various semantic description methods and justified the choice of translational description as the method of choice for the ITPSL semantics. Translational semantics are all about code generation. In the case of ITPSL, a code generator specifies how the ITPSL symbols will be translated to LUARM [82] SQL statements.

Compilers [134] are software programs that perform the code generation. There are various approaches for building a code generator. During the early days, most code generation was hard coded. This means that the logic of the software generator would implement operations to parse the input language, validate its syntax and then transform the statements into the target language according to pre-set rules that were written by hand. This early approach required a lot of work for complex code generation and was rather inflexible.

More recent approaches introduce more flexibility by viewing a code generator as a model transformation formalism. The 'Query/View/Transformation' QVT formalism [121] is a characteristic example of this type of code generation approach. A generic tool takes as input the source language symbols as well as a set of model transformation rules and produces the symbols of the target language. Figure 6.5 displays a simple example of a QVT rule that converts book data to a standard publication reference format



```
metamodel BOOK {
  class Book {title: String; composes chapters: Chapter [*];}
  class Chapter {title : String; nbPages : Integer;}
}

metamodel PUB {
  class Publication {title : String; nbPages : Integer;}
}

transformation Book2Publication(in bookModel:BOOK,out pubModel:PUB);

main() {
  bookModel->objectsOfType(Book)->map book_to_publication();
}

mapping Class::book_to_publication () : Publication {
  title := self.title;
  nbPages := self.chapters->nbPages->sum();
}
```

Figure 6.5: Query/View/Transformation rules [121]

However, QVT formalisms are based on operational semantics [121]. The previous section discussed the need for referential transparency and thus adopting a QVT formalism for generating the LUARM SQL statements is not a good option. However, there is a good equivalent that will avoid the need for hard coding and will also satisfy the referential transparency of the functional programming paradigm.

The Extensible Markup Language (XML) [122] appeared as a W3C recommendation in 1998. Since then, XML has enjoyed considerable popularity as an interoperability solution for data exchange amongst heterogeneous computer systems.

In essence, XML is a syntax for creating a ‘markup’. A ‘markup’ is a set of elements, attributes, and other structures that facilitates meaningful labeling of data so that other human beings or software can understand and interpret information. The combined human and machine readability features of the XML markup and the well formed standardized data description rules it imposes [122] make it ideal for usage in ITPSL signatures for a number of reasons:

- XML's declarative syntax and tag naming flexibility make the process of creating various types of insider data easy.
- XML's hierarchy imposes a universal well agreed structure on how the information is presented and interpreted.
- XML standards such as that of XSLT [131] can provide a good basis for referential integrity, as discussed in section 6.2. In fact, XSLT can be a fully fledged programming language having a lot of common attributes with functional programming [123]. Its declarative orientation provides a good ground for making translational semantics.

In addition, reliable XML parsers (tools that extract data from XML semantics in a consistent fashion) are readily available in a variety of programming languages. This could save a lot of effort in the process of designing a language schema that could be parsed easily. Thus, if one is about to embed a DSL grammar in an XML markup, he could gain many advantages over a non-XML markup: the ability to have embedded well formed rules that are standard and can represent hierarchical information and relative ease of developing parsing routines on well tested software. Later paragraphs of this section will elaborate on how this applies to ITPSL.

Software developers have criticized XML's characteristics. No language design decisions should be dictated by implementation details, especially when there are arguments against using XML to design DSLs. While there are no guidelines on when to employ XML in relation to DSL construction, Bell [124] notes some overheads of XML when it comes to the excessive tag space occupied in relation to expressing simple data such as non hierarchical name-value pairs and doubts about the feasibility of XML enabling generic programming language constructs such as loops and conditionals. Moreover, many DSL or generic programming developers cast doubt over the general human readability of XML [125], as well as XML performance [126]. The following paragraphs will address these concerns in relation to the ITPSL requirements.

Starting with the issue of using XML to make a DSL, Bell's criticisms about the excess disk space are valid. However, they become an important hinter on data intensive paradigms. ITPSL signatures will range on average around 2 Kilobytes in size. If one multiplies 2 Kbytes by an estimated couple of thousand signatures that might be applicable for an IT infrastructure, we get 4 Gigs of signature data. In the Tbyte era of cheap disk storage and with various portable media been able to hold many times that amount of space, this is not a problem for our domain.

Bell [124] also addresses the complexity of building generic programming language features such as loops and conditional statements in XML schemata. This is also a valid criticism. XML was not designed to incorporate features of generic programming languages, as it is primarily optimized for the expression of declarative rules. Of all the generic programming features, ITPSL will need to implement conditionals and binary operators, in order to satisfy the functional high level requirement of representing decision theoretic information, as stated in section 4.3. This is a necessary overhead implemented in ITPSL by means of special XML tags such as <ifexists>, <AND> and others. Later sections of this chapter will present these semantic details.

Human readability is an important factor for the design of ITPSL semantics, as they are also intended for human consumption besides being a target for code translation by a code generator. The discussion on the human readability of XML [126] is contested. Human readability is a strong plus for every kind of programming language (generic and DSL). The XML readability issue seems to center around the fact that the mark-up overwhelms the data content in between, and “line after line of opening tag almost next to the identical closing tag is hiding the essential data by their redundancy”, as noted by Bos [127]. Whilst this is true, it does not mean that XML is completely unreadable. Some effort is required for reading XML statements. The usage of meaningful tag names that comes with the freedom of being able to create them reduces that effort and makes human readers be able to interpret the document easily, even without being familiar with the tag structure [128]. In addition, various XML editors and tools are able to display XML information in a more structured manner to aid the understanding of the information conveyed in the XML document. Thus, we conclude that although reading XML pages requires some effort by human readers, this is a small price to pay in order to have a well formatted descriptive structure as the basis for ITPSL.

Finally, the issue of performance raised in [126] does not really affect ITPSL. The ITPSL signature is not the basis of computation for the language to work. The signature holds data to express the basic elements of a particular threat. The data will have to be parsed and used in various modules to check for the type of threat, but the basis for computing the checks is not XML based. Consequently, any overhead encountered by the XML parser extracting the signature data is small compared to the non-XML based computational overhead of making sense of the data.

In support of XML, it is also worth emphasizing that a number of security specification languages are based on it. The specification for Extensible Configuration Checklist Description Format (XCCDF) [129] and the eXtensible Access Control Markup Language (XACML) [130] are two examples of research work that encapsulate their semantics in XML markup.

A final comment in favor of XML concerns three of its key technologies that relate to the ITPSL structure and function:

- The Extended Style Language Transformations (XSLT) standard [131]: A technology that transforms XML documents to any other type of document.
- The XML Path Language (XPath) standard [132]: A mechanism for selecting precisely parts of XML documents by specifying their exact location inside an XML document.
- The XML Schema standard [133]: An XML schema defines the structure, content and semantics of XML documents. As such, it can be a mechanism for validating the semantic structure of documents.

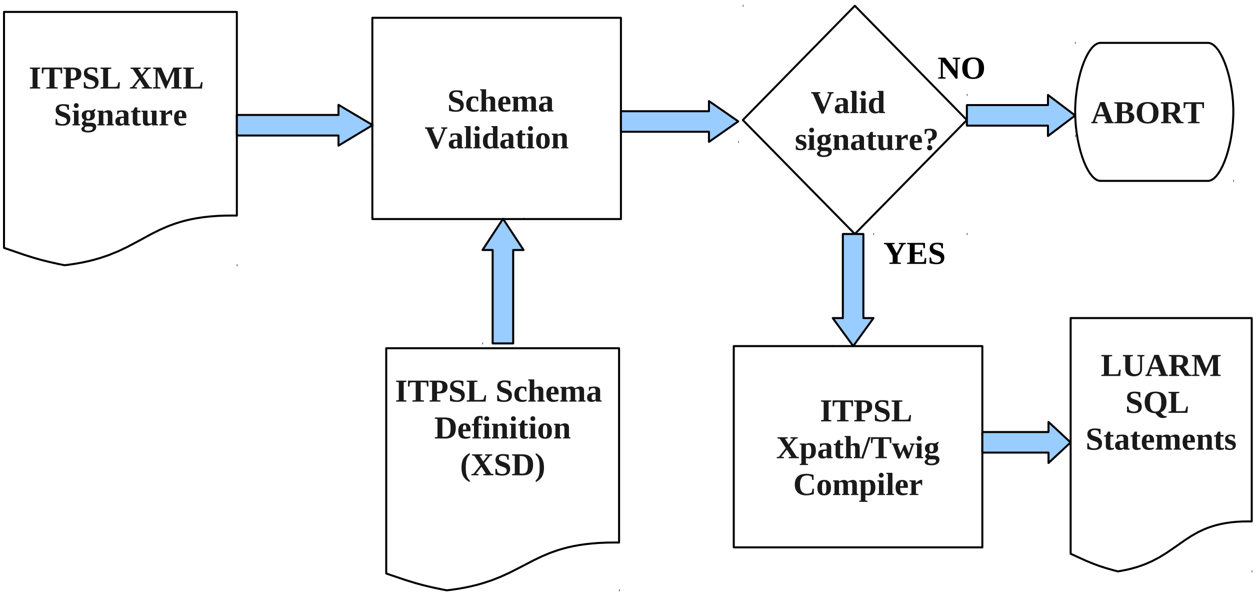


Figure 6.6: ITPSL XML components

Figure 6.6 above illustrates how these XML technologies are used in the information flow of the ITPSL 'compiler'. The ITPSL compiler will accept an XML based signature and will generate as output a set of LUARM based SQL statements, in order to mine relevant audit data, as described in Chapter 5. Letting the issue of the language semantics to the side for the moment, we assume that XML will provide the ITPSL structure. We can then show the multiple advantages of using an XML based approach over traditional approaches of constructing compilers.

On the top left, we have an XML document that contains the threat signature. The XML standard can guarantee the well-formed features of the markup [122] but not the syntactic validity of the ITPSL statements. If we were to build up a compiler, we would have to devise a regular grammar, build a lexer and tokenizer [134] and then write the parser routines, in order to make sense of the input file and infer its syntactic validity. This process can be cumbersome and time consuming. In contrast, this is a relatively easy process with the XML Schema standard [133]. The tool can reference an XML Schema Definition file (XSD), where the language author can encode (in XML) the syntactic rules and form. The XSD file and the input ITPSL threat signature are then fed into widely available XML Schema Validator modules. Such a module can reliably infer the syntactic validity of the signature file. Writing XSD files has of course its own learning curve, but it is a much simpler process than writing various compiler sub-modules from scratch.

After this initial pre-processing step, the syntactically valid ITPSL signature file is based on the actual ITPSL compiler. At this point, the XPath standard [132] and the Perl oriented XML::Twig module come [135] into the picture. This helps the compiler parse the ITPSL signature document by pointing at specific parts of the XML document, extracting the data and initiating actions depending on the data content. The actions are constructed LUARM SQL statements, bridging the gap between language semantics and audit data mining.

Appendix B provides sample code for both the XSD and the implemented prototype of the ITPSL Compiler, which is described in detail in Chapter 7 of the thesis.

For all these reasons, XML is a good standard to base the semantics of ITPSL, trading off a small part of the signature's human readability for having consistent structure and easiness in the construction of the ITPSL compiler. The next section will present the ITPSL language semantics.

6.4 An introduction to the ITPSL markup

The previous chapter sections discussed the programming paradigm and the markup basis for constructing ITPSL. It is now time to present the semantic framework of ITPSL, the core issue of this research project.

At the heart of the semantic framework lies the form of the ITPSL signature, a semantic structure that represents the encoding of an insider threat. Figure 6.7 shows the general structure of an ITPSL signature. It consists of a header section followed by the main body of the signature where sub-blocks of file, exec, network and hardware statements are encapsulated, in accordance to the different types of LUARM audit log data.

All ITPSL signature sections conform to the XML well formed rules [122] and a specific XML schema [133] against which are validated. Appendix B contains a suitably crafted XML Schema Definition file (XSD) that encodes the ITPSL content and syntactic rules. The rest of this section describes what constitutes a valid ITPSL signature in detail and cross-referencing language features with the ITPSL functional requirements of section 4.3 .

The ITPSL header section provides important signature classification information. This is in response to ITPSL functional FR2 (building of signature repositories). The <signid> tag marks the signature’s unique identification number. At the time of signature creation, the system takes the current date and time combined with some other semi random metrics and creates a unique MD5 hash [91], in order to make a unique signature ID. The <signdate> section is self-explanatory.

```
<itpslsig>
<itpslheader>
        <signid> <md5sum of date and second, type of OS, current number of processes>
        </signid>
        <signdate>
                <year> dddd </year>
                <month> dd </month>
                <day> dd </day>
        </signdate>
        <ontology>
                <reason> “intentional” | “accidental” </reason>
                <revision> d.d </revision>
                <user_role> “admins” | “advanced_users” | “ordinary_users” </user_role>
                <detectby>  “file” | “exec” | “network” | “hardware” | “multi” </detectby>
                <multihost> yes | no </multihost>
                <hostlist> host1,hostgroup1,...hostn,hostgroupn </hostlist>
                <weightmatrix>$n_{events}$, $w_{event1}$,$w_{event2}$,...,$w_{eventn}$ </weightmatrix>
                <os> “linux” | “windows” | “macosx” | “unix” </os>
                <osver> “2.4” | ”2.6” |  “2000” | “Vista” | “7” </osver>
                <threatkeywords> keyword1 keyword2 ... keyword5
                 </threatkeywords>
                [ <synopsis> “text that describes the signature’s purpose and function”
                </synopsis>]
        </ontology>
</itpslheader>
<itpslbody>
        <mainblock>
                <mainop> “AND”|”OR”|”XOR”|”as_a_result_of” | “justone”</mainop>
                <subblock>
                        <subop> “AND”|”OR”|”XOR”|”as_a_result_of”| “single” </subop>
                         <filestatements> ….</filestatements>
                         <execstatements>….</execstatements>
                        <netstatements>…</netstatements>
                </subblock>
                <subblock>
                        <subop> “AND”|”OR”|”XOR”|”as_a_result_of”| “single” </subop>
                         <filestatements> ….</filestatements>
                         <execstatements>….</execstatements>
                        <netstatements>…</netstatements>
                </subblock>
        </mainblock>
</itpslbody>
</itpslsig>
```

Figure 6.7: General ITPSL signature structure

The ontology header subsection is of particular importance for the creation of signature repositories. An ontology is a data model [136] that represents a set of concepts within a domain (or formally known as domain of discourse in linguistic terms [137]) and the relationships between those concepts. A variation of the threat model published by Furnell and Magklaras [56] and the audit engine data constitute the data model and the domain that the proposed markup language addresses.

As discussed in section 4.3, one of the problems ITPSL is trying to address is the lack of insider threat scenario repositories . When such a repository is constructed, facilities to search and relate signatures (descriptions of threat scenarios) will be important and thus the language must have both semantic and data identifiers to allow security specialists to locate a class of signatures. For example, one could select all signatures that use network detection criteria, or all signatures that target p2p client installation and detect their presence at multiple levels ('multi' refers to a combination of employing 'file', 'exec', 'network' and 'hardware' detection statements). A third example could be the last two revisions of a particular set of signatures or a set of signatures whose weight matrix places more emphasis on network detection criteria.

The <reason> tag specifies whether we are searching for a threat that is a result of deliberate actions (intentional) or accidental mistakes (accidental). The <revision> tag makes possible to trace signatures whose detection criteria are modified to improve the accuracy or to examine slightly different aspects of the target problem. In that case, the ‘signid’ identifier remains the same amongst the related signatures.

The <multihost> and <hostlist> tags bind the signature to a particular host or group a host in two ways:

- **No cross-host correlation binding**: A <multihost> no </multihost> value combined with one or more hosts in the <hostlist> tag will apply the signature in each of the specified hosts (or host groups), looking for events on a single host at a time. This way should be used when we wish to bind the signature specification on one or more computing hosts and all the events concern the single specified hosts.
- **Cross-host correlation binding**: A <multihost> yes </multihost> value combined with two or more hosts in the <hostlist> tag will apply the signature across multiple hosts. This signature binding facilitates events that occur in multiple hosts, satisfying the ITPSL requirement FR8 (event correlation over multiple hosts). In such a case, each individual ITPSL sub-block event definition should specify the <onhost> tag, in order to bind the event to one or more of the specified hosts.

ITPSL uses a modified version of the weight matrix concept [56] discussed in section 4.3 of the thesis. This modification allows a signature author to further tune the sensitivity of the ITPM model [56], and is included in the signature ontology header (<weightmatrix> tag). There are two important operation modes defined by the content of the weightmatrix attribute in an ITPSL signature:

- **Detection mode**: This mode forces the signature to be interpreted as a way to detect an event. It is triggered by a weightmatrix directive of **<weightmatrix> 0 </weightmatrix>.** This indicates that the signature will produce a true or false result on the basis of whether the criteria described in the ITPSL body are met or not.
- **Predictive mode**: Any non zero weightmatrix directive (example **<weightmatrix> 2,45,55 </weighmatrix>**) triggers the interpretation of the signature in predictive mode. This means

that the signature will be interpreted in a way that will provide a likelihood of the occurrence of the described IT misuse incident.

The original version of the weight matrix concept [56] allowed the signature author to express his confidence on the various metrics and the nature of the incident he is trying to predict. The author would use a parenthesized list of nine digits, encoding the relative weights of the various ITQAs, as listed in Table 1 of section 4.3. This approach is a theoretical way that emphasizes the importance of grading the different threat qualifiers by using different categories of operating system data (file, network, process execution and attributes), but it can be inflexible. For instance, consider a signature that consists of network criteria only. In such a case, it is not clear how the weight distribution should occur in the Weight Matrix.

In order to resolve this issue and make the weight matrix concept more flexible for the encoding of insider threat signatures, the weight matrix should have the form of Figure 6.8. The **Evaluated Potential Misuse Occurrence (EPMO)** is defined as the sum of the weights of all the events encoded in the signature. In ITPSL, an event is a discrete specification of a system-level act (file, network, process execution, hardware device attachment) dictated by the LUARM audit record format. Events are described inside an ITPSL sub-block (<subblock></subblock>). Later paragraphs will discuss the context, structure and role of subblocks, when we examine the main body of the ITPSL signature. Hence, the content of the weightmatrix tag consists of a series of numbers: The first of them describes the number of events and the rest of the numbers are the weights for each event subblock. This adaptation of the Weight Matrix concept is a more flexible way to express the confidence of certain actions for threat prediction purposes and satisfies ITPSL requirement FR5 (encoding of decision theoretic information).

$$\sum \mathbf{wevent_n = EPMO}$$

EPMO -> Evaluated Potential Misuse Occurrence (0...1)
n-> number of specified events

<weightmatrix>nevents, wevent$_1$**,wevent**$_2$**,...,wevent**$_n$ **</weightmatrix>**

Figure 6.8: Weight Matrix in ITPSL

```
<itpslsig>
<itpslheader>
        <signid> 69754c2b65627a098d02eb6244e40e69 </signid>
        <signdate>
                <year> 2007 </year>
                <month> 08 </month>
                <day> 25 </day>
        </signdate>
        <ontology>
                <reason> intentional </reason>
                <revision> 1.0 </revision>
                <user_role> ordinary_users </user_role>
                <detectby> multi </detectby>
                <multihost> no </multihost>
                <hostlist> cn1.abc.com </hostlist>
                <weightmatrix> 2,40,60 </weightmatrix>
                <os> linux </os>
                <osver> 2.6 </osver>
                <threatkeywords> p2p installation azureus </threat>
                [ <synopsis> “This signature estimates the threat of installing and using the
                azureus p2pclient” </synopsis>]
        </ontology>
</itpslheader>
<itpslbody>
.....
</itpslbody>
</itpslsig>
```

Figure 6.9: An example ITPSL signature header

Figure 6.9 provides an example of an ITPSL signature header (the signature main body is excluded) made in August 2007, targeting an intentional misuse act that concerns the installation of an azureus p2p client by the 'ordinary_users' category for the Linux operating system. The weightmatrix informs that there are two event definitions (thus two subblocks in the signature), with relative weights of 40 and 60 respectively.

In summary, the signature header provides threat signature metadata and helps the ITPSL signature author to create signature repositories, as well as to apply the signature to particular computing devices. We can now move to the ITPSL body (<itpslbody></itpslbody>), the main part of the signature where the encoding of the insider threats takes place.

Figure 6.10 displays the structure of the signature body (the header and the rest of the markup is excluded for illustration purposes). The ITPSL body contains one main statement block (<mainblock>). Each of the file, process execution, network and hardware statements are contained within statement sub-blocks (located always inside the main block) whose start and end are marked with the <subblock> tag.

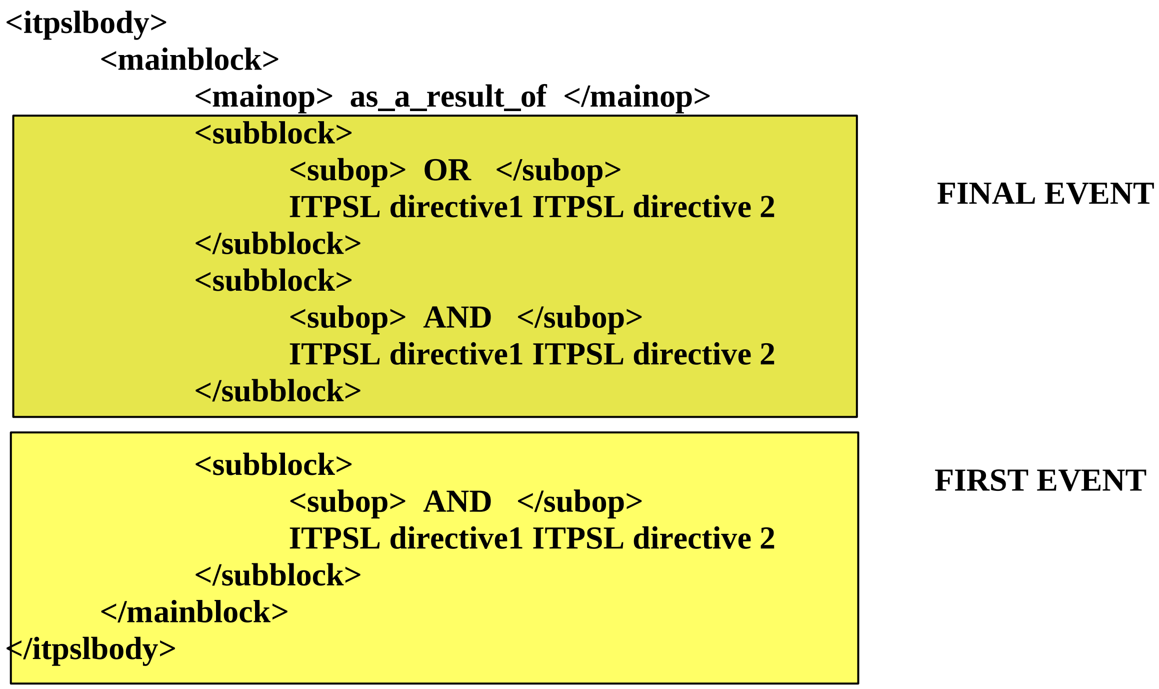
Previous paragraphs have already explained that a sub-block defines an event, a discrete specification of a system-level act, as recorded by a LUARM audit record. The various types of sub-blocks and their encapsulating statements can be combined to create complex scenarios by using the <mainop> and <subop> tags. These tags act as logical operators. The <mainop> tag applies the AND OR XOR, as well as the “justone” operator, in order to define how to consider the content of various sub-blocks. These operators provide the foundation to express various aspects of ITPSL requirement FR4. The <subop> tag does the same job for the statements contained inside the scope of a sub-block. The inclusion of these logical operators in the language aims to enable the description of various potential scenarios (polymorphism), as dictated by requirement FR5 (Section 4.3).

```
<itpslbody>
        <mainblock>
                <mainop> “AND”|”OR”|”XOR”|”as_a_result_of” | “justone” </mainop>
                <subblock>
                        <subop> “AND”|”OR”|”XOR”|”as_a_result_of”| “single”      </subop>
                         <filestatements> ….</filestatements>
                         <execstatements>….</execstatements>
                        <netstatements>…</netstatements>
                 </subblock>
                <subblock>
                        <subop> “AND”|”OR”|”XOR”|”as_a_result_of”| “single”      </subop>
                         <filestatements> ….</filestatements>
                         <execstatements>….</execstatements>
                        <netstatements>…</netstatements>
                 </subblock>
        </mainblock>
</itpslbody>
```

Figure 6.10: The ITPSL signature body

The ‘as_a_result_of’ operator is an operator that enables the expression of sequenced events. Its scope is always inside the mainop block. The event sequence description provides the ability to capture a sequence of events in order to increase the confidence of detecting the right conditions for the threat the signature addresses (ITPSL requirement FR4). Figure 6.11 provides an example of this feature.

What this description expresses is that the events described by the three sub-blocks should be captured by the audit engine one after the other. The first of the listed sub-blocks represents the final event. The very last sub-block describes the first event. This semantic structure was chosen, in order to make the first specified event appear as the consequence of all the previous events and enhance the relation between the steps that constitute a threat scenario. The same holds true for the subop operator inside each sub-block.



```
<itpslbody>
        <mainblock>
                <mainop>  as_a_result_of  </mainop>
                <subblock>
                        <subop>  OR   </subop>
                        ITPSL directive1 ITPSL directive 2
                </subblock>
                <subblock>
                        <subop>  AND   </subop>
                        ITPSL directive1 ITPSL directive 2
                </subblock>
                <subblock>
                        <subop>  AND   </subop>
                        ITPSL directive1 ITPSL directive 2
                </subblock>
        </mainblock>
</itpslbody>
```

Figure 6.11: The 'as_a_result_of' operator (main block scope)

There are four broad categories of threat detection statements (ITPSL directives), reflecting the core based consequences of the Magklaras and Furnell insider misuse taxonomy [36] and the respective audit record types of LUARM: file statements, execution statements (originally referred to as memory manipulation in the taxonomy) network statements and hardware statements. File statements provide the ability to describe aspects of threats that relate to operating system file operations. The execution statements help the signature author to describe threat elements associated with the execution of programs. The ‘netstatements’ are designed to facilitate threat prediction by looking for signs of certain network activities. Finally, hardware statements describe the attachment of hardware devices to the system. The inclusion of these four basic types of statements in the ITPSL semantics mixes the ability to specify static and live forensic data as dictated by ITPSL requirement FR3.

```
<itpslsig>
<itpslheader>
        …
        <detectby>file</detectby>
        <weightmatrix>3,50,20,30</weightmatrix>
        …
</itpslheader>
<itpslbody>
        <mainblock>
                <mainop> AND </mainop>
                <subblock>
                        ....
                </subblock>
                <subblock>
                        ....
                </subblock>
        </mainblock>
</itpslbody>
</itpslsig>
```

Figure 6.12: Example of an invalid ITPSL signature

The various types of ITPSL statements can be entered in any order and not all of them are obligatory. Signature authors should also know that there needs to be consistency between the content of the header and body ITPSL parameters. For example, if one specifies two sub-blocks in the main body and three event weights in the Weight Matrix header tag, the signature will be invalid (Figure 6.12).

Beyond the ITPSL XML Schema validation that dictates the syntax and content of the language markup, a number of additional validity issues are checked by the ITPSL compiler, as part of the signature's validation procedure. If an inconsistency is found, the ITPSL signature parser will reject the signature and output a suitable error message to flag the problem. Further examples of valid and invalid signatures will be given throughout the text of the next chapter sections.

6.5 ITPSL file and directory statements

Having introduced the basic structure of the proposed language markup, this section now outlines the ITPSL file and directory statements. In most operating systems, directories are implemented as special file structures [42] [43]. Nevertheless, separate directory statements exist in the language because the context of expressing insider misuse actions differs between files and containers of files (directories). The file and directory statements provide the important mechanism of specifying file related activities. Previous chapters indicated that many misuse activities have file related system consequences. Thus, a misuse language must be able to express misuse scenarios by means of looking at filesystem related information. There are three broad categories of file statements:

- **File presence detection statements (fileexists, direxists)**: These statements help the language user to relate the presence of certain files and directories to misuse scenarios. They also represent the workhorse of the ITPSL file statements as they are re-used (implicitly or explicitly) by other types of statements (for example, a file access specification statement requires that we check for the file presence).
- **File access ability statements (usercanaccessfile, usercanaccessdir, groupcanaccessfile, groupcanaccessdir)**: These represent a mechanism that relates the ability of certain individuals to access file data to misuse threat scenarios. This is important for an insider misuse specification language because it allows decision theoretic information to be expressed in a misuse signature. The fact that certain file data can be accessed does not necessarily represent misuse, but it could indicate the possibility for certain types of misuse (information theft, etc). All well known operating systems place great emphasis on file access control operations by employing various forms of Access Control Lists (ACLs) [1]. There are discre-

tionary, mandatory and role-based ACLs that constitute different approaches on controlling file access amongst operating systems. However, the important thing in the process of specifying a threat scenario is to focus on what can be accessed and by whom and express this info using standardized semantics. The ITPSL engine should be aware of the differences of the various ACL mechanisms and hence it should be able to examine whether the conditions the language semantics specify are true or false.

- **File access statements (fileaccess, diraccess)**: A file access statement helps the language user to specify relevant file activities that should be considered as part of a misuse signature. In contrast to the two previous types of file statements that examine file presence or access capability, this type of statement refers to the actual act of file-system access. In addition, whilst file presence and access ability statements express facts without a specified time period, a file access statement specifies either current (at the time of the misuse signature check) or other time-specific access activities, enhancing the ability of the language to relate temporal information to threat scenarios.

6.5.1 File and directory detection statements

The 'fileexists' detection statement makes sure that the signature author can locate one or more files (see <singlefile></singlefile> tags) that meet specific conditions. It is the workhorse of the file operation statements and thus it needs to balance specificity and flexibility of expression to accommodate a wide range of scenarios. Figure 6.13 shows the basic syntax of the statement.

The filename tag tries to locate files with specific names. The specified name must include the file extension. If that fails or a specific name is not entered, the entered name becomes part of a regular expression that the ITPSL engine will produce in an attempt to locate the file(s) in question. If the entered name fails to intercept a file as a regular expression (many malicious acts tend to alter the name of files to hide them or trick users, including the file extension), then the rest of the criteria are employed to detect a file. The AND, OR, XOR and NOT logical operators may be used to specify many combinations of matching file names.

```
<fileexists>
        <filename>  AND|OR|XOR|NOT (name1, name2, name3…) </filename>
        <type> AND|OR|XOR|NOT (executable | image | sound | video | textdata | special | any)
</type>
        <location> path | AND|OR|XOR|NOT (path1, path2) | #userhome# | any   </location>
        <singlefile> yes | no </singlefile>
        [
        <quantity> (over|less|equal)  number </quantity>
        <filesize> (over|less|equal) number (b/k/m/g) </filesize>
        <withcontents>
                <AND|OR|XOR|NOT>
                        <stringsearch> “string”</stringsearch> | <hexsearch> “hexstr”
                                </hexsearch> |
                        <stringsearch> AND/OR/XOR/NOT (“string1”,”string2”…)
                </stringsearch> |
                        <hexsearch>  AND/OR/XOR/NOT (“hexstr1”,”hexstr2”…)</hexsearch>
                </AND/OR/XOR/NOT>
        </withcontents>
        <withchecksum> “md5checksum” </withchecksum>
        <ownedbyuser>username</ownedbyuser>
        <ownedbygroup>groupname</ownedbygroup>
        <ownerperm> (read,write,execute) </ownerperm>
        <groupperm> (read,write,execute) </groupperm>
        <otherperm> (read,write,execute) </otherperm>
        <specialperm> setuserid,setgroupid </specialperm>
        ]
</fileexists>
```

Figure 6.13: The 'fileexists' ITPSL statement

The <type> tag is employed to examine whether the file(s) in question are executable, text data or other special file (such as device driver or other inter-process communication files).

The <location> tag tells the ITPSL engine where to start looking for the file(s). A specific path or combinations of possible paths where the files in question reside might be entered. However, in every case, the specified path acts as a starting point for the file search, and the ITPSL engine will attempt to search in all possible subdirectories. This can affect the time it takes to execute the search operation. Large filesystems might contain millions of files under various subdirectories and can prolong the time and computational resources required for their execution. The author can choose the “any” value to instruct the engine to start searching from the top of the filesystem tree, specify certain filesystem paths or choose the ‘userhome’ value. This latter value will start the file search from the user’s home area. The signature’s header will target a particular user category (i.e. ordinary users) and thus every user’s home directory will be examined with the conditions specified in the statement.

The use of the <location> tag is obligatory in a standalone ‘fileexists’ statement. However, it can be omitted whenever a fileexists statement is embedded inside a direxists statement (see ‘direxists’ statement). This feature is necessary, in order to link the context of the embodied fileexists to that of the direxists statement and thus the path of the specified file is automatically resolved by the relevant tag values of the direxists statement. Nesting statements of the same category under certain rules, in order to increase the specificity of an expression is a language feature and subsequent paragraphs will explain the concept in more detail.

The <withcontents> optional statement allows one to specify files that have certain contents searcheable by general alphanumeric string or hexadecimal patterns. In case the search concerns a single file (<singlefile> yes </singlefile>), an md5sum checksum could be entered, so that the author is certain to intercept the presence of a specific file. On the other hand, if we are looking for many files, it might be useful to provide a threshold value for the number of the detected objects. This is the role of the <quantity> optional tag. It gives the signature author the ability to set a threshold number of detected files and connect it to the target scenario in the header of the signature.

As an example, the following statement detects a specific image file with name 'supernova1.jpg' under the directory /storage/images (Figure 6.14).

```
<fileexists>
        <filename>  supernova1.jpg </filename>
        <type> image </type>
        <location> /storage/images </location>
        <singlefile> yes  </singlefile>
        <withchecksum> ca12b5405dfb739e197c803ed790bc01</checksum>
</fileexists>
```

Figure 6.14: Example of detecting the presence of a specific image file

This file existence statement of Figure 6.15 demonstrates a search for multiple mp3 files under the user's home directory, setting a threshold of over 100 files. This could be part of a multi-level signature trying to detect the possibility of illegal peer-to-peer software activity.

```
<fileexists>
        <filename> *.mp3 </filename>
        <type> audio </type>
        <location> userhome </location>
        <singlefile> no </singlefile>
        <quantity> over  100 </quantity>
</fileexists>
```

Figure 6.15: Example of detecting the presence of a number of music files

Despite the existence of optional statements parts, increasing the specificity of the criteria employed in the file existence statement will improve its effectiveness.

In most known operating systems, directories are containers of files, so from a file operations point of view it is important to have the ability to relate actions and events to directories. The ‘direxists’ file statement facilitates directory searching and like ‘fileexists’ contains tags that set the conditions for choosing the directories.

```
<direxists>
        <dirname>  AND/OR/XOR/NOT (name1, name2, name3…) </dirname>
        <location> AND/OR/XOR/NOT (name1, name2, name3…) | userhome | any
</location>
        <singledir> yes | no </singledir>
         [
        <quantity> (over|less|equal)  number </quantity>
        <dirsize> (over|less|equal) number (b/k/m/g) </dirsize>
        <withfiles>
                <AND|OR|XOR>
                        <fileexists>….</fileexists>
                        <fileexists>….</fileexists>…
                </AND|OR|XOR>
        </withfiles>
        <ownedbyuser>username</ownedbyuser>
        <ownedbygroup>groupname</ownedbygroup>
        <ownerperm> (read,write,execute) </ownerperm>
        <groupperm> (read,write,execute) </groupperm>
        <otherperm> (read,write,execute) </otherperm>
        <specialperm> setuserid,setgroupid </specialperm>
]
</direxists>
```

Figure 6.16: The ITPSL 'direxists' statement

Similar to a ‘fileexists’ statement, ‘direxists’ checks for the presence of directories. Most of the tags should be self explanatory as they define properties similar to those of the ‘fileexists’ statement. However, entire ‘fileexists’ statements can be included inside an optional <withfiles></withfiles>

block, in order to increase the specificity of a directory search operation. Within the <withfiles> tag, the signature author can place any relevant file statement(s) and expect the encapsulated ‘fileexists’ statement(s) to act as selection condition(s) of the ‘direxist’ statement. Statement nesting can be performed under certain conditions only for statements of the same type. For instance, if we have a file statement such as ‘direxists’ and its syntax dictates that between the ‘withfiles’ tags other file statements may be placed, this will be true within the domain of file statements. For example, one cannot nest a net statement or exec statement inside a file statement.

```
<direxists>
        <dirname>  Filme </dirname>
        <location> userhome </location>
        <singledir> yes </singledir>
        <dirsize> over 2 g </dirsize>
        <withfiles>
                <AND>
                        <fileexists>
                                <filename> *.torrent </filename>
                                <type>  textdata </type>
                                <singlefile> no </singlefile>
                                 <quantity> over 20 </quantity>
                        </fileexists>
                        <fileexists>
                                <filename> OR (*.avi, *.wmv, *.mpg)  </filename>
                                <type> video </type>
                                <singlefile> no </singlefile>
                                <quantity> over 20 </quantity>
                        </filexists>
                </AND>
        </withfiles>
</direxists>
```

Figure 6.17: Example of detecting the presence of a specific directory

As an example of a ‘direxists’ statement, one can consider a scenario that requires the detection of a directory under the user’s home area that is called ‘Filme’. That directory should be more than 2 Gigabytes in size and contain more than 20 torrent files, as well as 20 or more video files of either avi, wmv or mpg type. This directory detection scenario can be expressed with the statement shown in Figure 6.17.

Notable is the absence of the <location> tag in the embedded ‘fileexists’ statements. In this case, the nested 'filexists' tags embed statements that increase the specificity of the directory we are trying to detect. The ITPSL engine will automatically construct the <location> tag of each embedded fileexists statement by merging the <location> and <dirname> tags values in order to form a suitable file path. Thus, in this case, all file searches will be performed under the 'Filme' directory of the user’s home area.

6.5.2 File access ability statements

An insider threat prediction specification language must be able to provide the ability to express potential file access scenarios, in order to support decision theoretic information. Even if the user is not accessing the file(s) in question, knowing whether a file is accessible by a user, as well as what the user can do with this file can form the basis of assessing various threat scenarios.

The ‘usercanaccessfile’ statement provides this capability (Figure 6.18). It ties the existence of a file to the ability of a user to read, write and/or execute the file. The similarity of its parameter tags to the ones of the ‘fileexists’ statements implies that the ITPSL engine will check that the files exist prior performing the access checks.

```
<usercanaccessfile>
        <userid>username</userid>
        <filename> AND/OR/XOR/NOT (name1, name2, name3…)  </filename>
        <accessrights> read| read-write| read-execute |all </accessrights>
        <type> AND/OR/XOR/NOT (executable | image | sound | video | textdata | special |
any) </type>
        <location> path | AND/OR/XOR/NOT (path1, path2) | userhome | any  </location>
        <singlefile> <yes | no> </singlefile>
        [
        <quantity> (over|less|equal)  number </quantity>
        <filesize> (over|less|equal) number (b/k/m/g) </filesize>
        <withcontents>
                <AND|OR|XOR|NOT>
                        <stringsearch> "string"</stringsearch> | <hexsearch> "hexstr"
</hexsearch> |
                        <stringsearch> AND/OR/XOR/NOT ("string1","string2"…)
                </stringsearch> |
                        <hexsearch>  AND/OR/XOR/NOT ("hexstr1","hexstr2"…)</hexsearch>
                </AND/OR/XOR/NOT>
        </withcontents>
        <withchecksum> "md5checksum" </withchecksum>
        <ownedbyuser>username</ownedbyuser>
        <ownedbygroup>groupname</ownedbygroup>
        ]
</usercanaccessfile>
```

Figure 6.18: The 'usercanaccessfile' ITPSL statement

The ‘groupcanaccessfile’ statement functions in a similar way to the ‘usercanaccessfile’ statement, however the checks are performed against a group of users. The word ‘group’ might refer to the notion of a file group, as implemented by the host operating system. For example, if the host operating system is UNIX, net groups are not taken into consideration. Whilst the exact mechanism of file groups is operating system dependent, all popular operating systems support the idea of file groups. The alternative is to specify a list of usernames against which the checks will be performed. In that case, the username list does not constitute an operating system group, but a logical group for the purposes of the language being able to check a set of criteria against users that might not belong to the same operating system group. In addition, this feature enables the language to specify logical groups in an operating system independent way.

```
<groupcanaccessfile>
<groupid>groupname | (username1,username2…)</groupid>
<filename>  AND/OR/XOR/NOT (name1, name2, name3…)  </filename>
<accessrights>”read” “read,write” “read, execute” “all”</accessrights>
<type> AND/OR/XOR/NOT (executable | image | sound | video | textdata | special | any)
</type>
        <location> path | AND/OR/XOR/NOT (path1, path2) | userhome | any   </location>
        <singlefile> <yes | no> </singlefile>
        [
        <quantity> <over|less|equal> “ number“ </quantity>
        <filesize> <over|less|equal> <number b/k/m/g> </filesize>
        <withcontents>
<stringsearch> “string1” AND/OR/XOR “string2”… </stringsearch>
AND/OR
<hexsearch> “hexpattern1” AND/OR/XOR “hexpattern2”…</hexsearch>
        </withcontents>
        <withchecksum> “md5checksum” </withchecksum>
        <ownedbyuser>username</ownedbyuser>
        <ownedbygroup>groupname</ownedbygroup>
]
</groupcanaccessfile>
```

Figure 6.19: The 'groupcanaccessfile' ITPSL statement

For instance, if we wish to see whether a user has write access to the /etc/passwd file of a unix host, we could achieve that by placing the following file statement in the body of the ITPSL signature.

```
<usercanaccessfile>
        <filename>passwd</filename>
        <accessrights>read,write</accessrights>
        <type> textdata </type>
        <location> /etc </location>
        <singlefile> yes </singlefile>
</usercanaccessfile>
```

Figure 6.20: A simple 'usercanaccessfile' usage example

Figure 6.20 displays a more specific scenario, where we wish to see if the users of the ‘accounting’ file group have read access to a commercially sensitive image file called “prototype1.jpg”, instances of which could be placed under the /drawings or under the /data/machinedesigns directories, is owned by username ‘jdavies’ and has an md5 checksum of cf7af1a746f000a0e02cf8b4d1b71ba9 (so we can be sure we will be looking for the right file).

The ‘<singlefile>no</singlefile>’ statement implies that the engine can check for more than one instance of the file. If the file exists only in one of the specified locations, the check will be performed only in one instance. This could satisfy a scenario where the file in question is placed in different folders with the same or different permissions.

```
<groupcanaccessfile>
        <groupid> accounting </groupid>
        <filename>prototype1.jpg</filename>
        <accessrights>read</accessrights>
        <type>image</type>
        <location>/drawings OR /data/machinedesigns</location>
        <singlefile>no</singlefile>
        <withchecksum> cf7af1a746f000a0e02cf8b4d1b71ba9 </withchecksum>
        <ownedbyuser>jdavies</ownedbyuser>
</groupcanaccessfile>
```

Figure 6.21: A 'groupcanaccessfile' scenario

The directory related capability access statements work in a similar manner. Figure 6.21 displays the way of specifying the ability of a user to access a directory by using the ‘usercanaccessdir’ statement. The analyst needs to specify the id of the user, the name and location(s) of one or more directories, as well as the access ability to either read access the contents of the directory or whether the specified user has full control of the directory and its contents. In the latter case, this means that the user could create and delete files in the directory, as well as create files or subdirectories under it, with all the security implications this might entail. The remaining optional tags are similar to the ‘direxists’ statement and can be used to increase the specificity of the directory selection.

```
<usercanaccessdir>
    <userid>username</userid>
    <dirname>  AND/OR/XOR/NOT (name1, name2, name3…) </dirname>
    <location>  AND/OR/XOR/NOT  (name1,  name2,  name3…)  |  userhome  |  any
    </location>
<singledir> yes </singledir>
<ability> just-read | full </ability>
[
    <quantity> (over|less|equal)  number </quantity>
    <dirsize> (over|less|equal) number (b/k/m/g) </dirsize>
    <withfiles>
        <AND|OR|XOR>
            <fileexists>….</fileexists>
        </AND|OR|XOR>
    </withfiles>
    <ownedbyuser>username</ownedbyuser>
    <ownedbygroup>groupname</ownedbygroup>
]
</usercanaccessdir>
```

Figure 6.22: The 'usercanaccessdir' ITPSL statement

```
<groupcanaccessdir>
    <groupid>groupname</groupid>
    <dirname> AND/OR/XOR/NOT (name1, name2, name3…) </dirname>
    <location> AND/OR/XOR/NOT (name1, name2, name3…) | userhome | any
    </location>
    <singledir> <yes | no> </singledir>
    <ability> just-read | full </ability>
    [
    <quantity> (over|less|equal)  number </quantity>
    <dirsize> (over|less|equal) number (b/k/m/g) </dirsize>
    <withfiles>
        <AND|OR|XOR>
            <fileexists>….</fileexists>
        </AND|OR|XOR>
    </withfiles>
    <ownedbyuser>username</ownedbyuser>
    <ownedbygroup>groupname</ownedbygroup>
    ]
</groupcanaccessdir>
```

Figure 6.23: The 'groupcanaccessdir' ITPSL statement

The ‘groupcanaccessdir’ statement (Figure 6.23) is also similar to ‘usercanaccessdir’ but it examines the ability of a group of users to access in common one or more directories. As with the ‘groupcanaccessfile’ statement, the groupid tag specifies an operating system file group or a list of usernames. The function of the rest of the statement tags is identical to the ‘usercanaccessdir’ statement.

A common scenario in many popular operating systems is that certain server processes (called ‘services’ in Windows-based operating systems or ‘daemons’ in the Unix/Linux/Mac OS X world) are run with user specific credentials to ensure access compartmentalization. Good system administration practice dictates to check that these user credentials cannot access files on areas beyond their operational control, as part of accidental information theft measures. For instance, MySQL databases run usually with a default specific user-id of ‘mysql’. Thus, as an example of a ‘usercanaccessdir’ statement, a scenario that checks whether user ‘mysql’ has read access to a number of sensitive directories (/shared/data2, /shared/data3) could be expressed as shown in Figure 6.24.

```
<usercanaccessdir>
        <userid>mysql</userid>
        <dirname>  OR (data2, data3) </dirname>
        <location> /shared </location>
        <singledir> yes </singledir>
        <ability> just-read </ability>
</usercanaccessdir>
```

Figure 6.24: An example of the 'usercanaccessdir' statement

A more complex example (Figure 6.25) would check that a group of users should not be able to create or alter files on directories that certain files are present. The example below checks whether all usernames ‘nickb’, ‘johnb’, ‘katea’ can read, alter, create and erase files under two directory names that can be under the directory with names “plans” or “schemas”. The directory could be either under in two potential locations. Hence, the ITPSL engine will check for four possible directories:

(/data/plans, /repos/plans, /data/schemas and /repos/schemas). For each of the existing directories, it will attempt to see whether they have a specific jpg file and a number of documents, before they qualify. It will then perform the directory access checks against each of the listed usernames.

```
<groupcanaccessdir>
        <groupid>(nickb, johnb,katea)</groupid>
        <dirname> OR (plans, schemas) </dirname>
        <location> XOR (/data, /repos) </location>
        <singledir> yes </singledir>
        <ability> full </ability>
        <withfiles>
                <AND>
                        <fileexists>
                                <filename> rs3000schematic.jpg </filename>
                                <type>  image </type>
                                <singlefile> yes </singlefile>
                                <withchecksum> b87af1a746f345a0e02cb8b4d1b71b14
                                        </withchecksum>
                        </fileexists>
                        <fileexists>
                                <filename> OR (*.doc, *.docx, *.pdf)  </filename>
                                <type> textdata</type>
                                <singlefile> no </singlefile>
                        </filexists>
                </AND>
        </withfiles>
</groupcanaccessdir>
```

Figure 6.25: Checking for directory access rights with 'groupcanaccessdir'

6.5.3 File and directory access statements

The actual act of accessing a file (as opposed to the existence or the ability to access it) is expressed by a ‘fileaccess’ statement. The signature author can relate access to one or more files to a certain user or groups of users by means of employing the <byuser> or <bygroup> tags. It is possible that file access can be checked not against a specific user or group of users (‘anyone’,’anygroup’ values). The possible locations and file names are specified in the same way as the previous file statements. However, in addition to the two previous file statement categories, file access statements provide the capability of specifying patterns of file access. In particular, ‘fileaccess’ makes use of the <pattern> tag, in order to specify:

- **File access on a temporal basis**: When it is important to specify whether access is current (time of the signature check) or when access takes place between certain time periods of the present day, the day before, the present week or the present month.
- **File access on a temporal frequency basis**: When it is important to specify how often access takes place within a specified period of time.

Enabling ITPSL to express temporal file access patterns is important. The additional specificity added by employing mechanisms to specify temporal frequencies can enhance the applicability of decision theoretic information in the process of identifying a specific threat (requirements FR3, FR4, FR5, section 4.3). A file access event intercepted under certain conditions might be important. The recurrence of the same file event under the same conditions is more important because it can increase the confidence of a threat indication. Thus, any language assigned to the task of expressing threat indicators should be able to express temporal frequencies.

Figure 6.26 below displays the structure of the ITPSL 'fileaccess' statement. The <byuser> and <bygroup> tags associate the file access activity to certain user and user group entities, a vital issue that provides file operation accountability. It is also possible to distinguish between file read and write operations (<accessmode> tag). The rest of the fields should be self-explanatory and similar to tags used in previous file and directory ITPSL statements.

```
<fileaccess>
        <byuser>username | anyone </byuser> | <bygroup>groupname | anygroup</bygroup>
        <accessmode>read | write </accessmode>
        <filename>  AND/OR/XOR/NOT (name1, name2, name3…) </filename>
        <type> AND/OR/XOR/NOT (executable | image | sound | video | textdata | special |
any)    </type>
        <location> AND/OR/XOR/NOT (name1, name2, name3…) | userhome | any
</location>
        <singlefile> <yes | no> </singlefile>
        <pattern> AND/OR/XOR/NOT ( now| hh-hh today | hh-hh (x | (0-30) ) days ago
[(more-than|less-than) x times | every (minute | hour | (Sunday-Saturday))  ) </pattern>
[
        <quantity> (over|less|equal)  number </quantity>
        <filesize> (over|less|equal) number (b/k/m/g) </filesize>
        <withcontents>
                <stringsearch> "string1" AND/OR/XOR "string2"… </stringsearch>
                AND/OR
                <hexsearch> "hexpattern1" AND/OR/XOR "hexpattern2"…</hexsearch>
        </withcontents>
        <withchecksum> "md5checksum" </withchecksum>
        <ownedbyuser>username</ownedbyuser>
        <ownedbygroup>groupname</ownedbygroup>
        <ownerperm> (read,write,execute) </ownerperm>
        <groupperm> (read,write,execute) </groupperm>
        <otherperm> (read,write,execute) </otherperm>
        <specialperm> setuserid,setgroupid </specialperm>
        ]
</fileaccess>
```

Figure 6.26: The 'fileaccess' ITPSL statement

As an example of a 'fileaccess' statement, Figure 6.27 provides an example that checks to see whether each of the users ‘ianb’ and ‘georgen’ read more than twice the file ‘/engineering/field-repos/fielddata.mdb’ between the hours of 20 and 24 in the evening yesterday.

```
<fileaccess>
        <bygroup>ianb,georgen</bygroup>
        <accessmode> read </accessmode>
        <filename>  fielddata.mdb </filename>
        <type>  textdata  </type>
        <location> /engineering/fieldrepos </location>
        <singlefile> yes </singlefile>
        <pattern> AND (  20-24 1 days ago, more-than 2 times) </pattern>
</fileaccess>
```

Figure 6.27: ITPSL 'fileaccess' example

We provide some further examples to clarify the expression of access patterns. If one wishes to check whether the same file was checked exactly 5 times throughout the whole day a week ago, the following <pattern> tag should be written:

<pattern> AND (00-24 7 days ago, 5 times) </pattern>

Verifying that the file in question was accessed any number of times between 23:00 and 24:00 hours for the last 30 days (including the present day) could be expressed as:

<pattern> 23-24 0-30 days ago </pattern>

Finally, a periodic access of more than 5 times between the hours of 23 and 24 every Saturday within the last 3 weeks could be expressed as:

<pattern> AND (23-24 0-21 days ago, 5 times every Saturday) </pattern>

These are some of the examples of file access pattern specification granularity that the language provides.

The ‘diraccess’ performs the same role for directory access (Figure 6.28). Most of the statement tags are similar to those of the ‘fileaccess statement’ including the way to specify access patterns, however the <accessmode> tag can now take the potential values of ‘just-read’ and ‘full’, reflecting the abilities to just read the directory contents or have complete directory access, which means the ability to modify, create and erase files respectively. The latter mode is detected by the ITPSL engine when files are opened in write mode by the specified user and group or when files are detected in write mode in general (‘anyone’ , ‘anygroup’ values for the <byuser> and <bygroup> tags).

```
<diraccess>
        <byuser>username | anyone </byuser> | <bygroup>groupname | anygroup</bygroup>
        <accessmode> just-read | full </accessmode>
        <dirname> AND/OR/XOR/NOT (name1, name2, name3…) </dirname>
        <location> AND/OR/XOR/NOT (name1, name2, name3…) | userhome | any
</location>
        <singledir> <yes | no> </singledir>
        <pattern> AND/OR/XOR/NOT ( now| hh-hh today | hh-hh (x | (0-30) ) days ago
[(more-than|less-than) x times | every (minute | hour | (Sunday-Saturday))  ) </pattern>
        [
        <quantity> (over|less|equal)  number </quantity>
        <dirsize> (over|less|equal) number (b/k/m/g) </dirsize>
        <withfiles>
                <AND|OR|XOR>
                        <fileexists>….</fileexists>
                        <fileexists>….</fileexists>…
                </AND|OR|XOR>
        </withfiles>
        <ownedbyuser>username</ownedbyuser>
        <ownedbygroup>groupname</ownedbygroup>
        <ownerperm> (read,write,execute) </ownerperm>
        <groupperm> (read,write,execute) </groupperm>
        <otherperm> (read,write,execute) </otherperm>
        <specialperm> setuserid,setgroupid </specialperm>
        ]
</diraccess>
```

Figure 6.28: The 'diraccess' ITPSL statement

As an example of 'diraccess' usage, consider a scenario that requires the check of whether user ‘michaels’ has accessed in full, during the previous day, a directory with name ‘/home/users/lars/data1’, as shown in Figure 6.29 below.

```
<diraccess>
        <byuser> michaels </byuser>
        <accessmode> full </accessmode>
        <dirname> data1 </dirname>
        <location> /home/users/lars </location>
        <singledir> yes </singledir>
        <pattern> 00-24 1 days ago </pattern>
        <ownedbyuser>lars</ownedbyuser>
</diraccess>
```

Figure 6.29: An example of 'diraccess' usage

The <pattern> tag expresses access patterns, as described in the 'fileaccess' ITPSL statement. In fact, the pattern expression syntax remains relatively consistent throughout the entire range of ITPSL statements, in order to keep the language consistent. Minor variations do exist, as the ITPSL context of the statement type might have specific requirements. These variations will be highlighted in the following sections of the chapter.

This concludes the presentation of the ITPSL file and directory statements. The next section will focus on defining misuse activities at the network level.

6.6 ITPSL network statements

The network statements express misuse scenarios in terms of network activity detected at host level. A large number of user activities relate to network enabled applications. Thus, the ability to describe aspects of network related user activities is important to a threat specification language.

There are two important design philosophies that dictated the construction of the ITPSL network statements. The first is concerned with balancing expressiveness and computational overhead and the second is trying to keep the underlying ITPSL engine simple.

Balancing expressive power and its associated computational overhead means that the network statements are host-level level oriented. The host-level network detection orientation means that the network statement expression criteria focus more on network endpoint classification detection rather than the actual payload of network activity and tap into the transported data (In ITPSL context, the term endpoint refers to the entity on one end of a TCP/IP transport layer connection). This was a conscious decision for two reasons:

1. A variety of network protocols that encrypt their payload [100] make the task of extracting and analyzing network data a difficult or impossible process.
2. Even if the underlying protocols that shift data between network endpoints are unencrypted, the increasing speeds of the data networks challenge network intrusion detection and prevention systems by creating a substantial computational overhead.

If ITPSL can provide mechanisms to relate endpoint activity to user, application and file data, this could compensate for any threat context lost by the ability to tap into the payload of a network connection. For example, if a substantial element of a threat could be identified only by the contents of a file downloaded as a result of a network connection (the network connection identifiers might not provide a clue), it is more efficient (or viable) to try and identify the file once it has reached the hard disk rather than trying to tap to a secure FTP connection and identify parts of this file in network packets. As long as one is able to say “This file is downloaded as a result of user x accessing endpoints a and b”, one can identify the payload in question in a viable and efficient way. Subsequent paragraphs describing network statements will illustrate how one can make the relation and identify events of interest, as a result of network activity.

The second aspect of the ITPSL network statement design is that they focus on the TCP/IP protocol suite. The world of data network communication protocols has a much wider scope than TCP/IP and other communication protocol architectures such as SNA, OSI and a variety of other proprietary protocols constitute a more complex picture. However, it is fair to say that TCP/IP has grown to be the dominant architecture of network communication for several decades now. Consequently, ITPSL is concerned only with TCP/IP to keep the implementation complexity of the ITPSL engine low.

The categories of network statements reflect to a certain extent those of the file statements in that they can be divided into 3 broad categories:

1. **Network element detection statements (netinterfaceexists, routeexists)**: These statements describe the presence of certain network interfaces or routes and they can be embedded in other types of net statements to increase the specificity of network expressions.

2. **Network access ability statements (usercanaccessnet, groupcanaccessnet)**: They describe the ability to access network resources as file access ability statements do for files. However, while a file access ability statement checks the permission of files by looking into persistent filesystem data, a network access ability statement will need to actively check whether a user can reach a specified network resource. Widely employed operating systems do not place this kind of information into persistent data structures. Hence, the ITPSL engine must be actively involved and launch a process that checks the specified network access resource criteria using the user or group credentials. Later paragraphs will outline the consequences of this feature.
3. **Network access statement (netaccess)**: This statement enables the language user to specify relevant network activities as part of a misuse signature (the equivalent of 'fileaccess'/'diraccess' for network operations). Pretty much like 'fileaccess', a 'netaccess' statement refers to the actual act of network access and it may also specify either current (at the time of the misuse signature check) or other time-specific endpoint access activities.

6.6.1 Network element detection statements

In ITPSL, a 'netinterface' is the logical entity associated with a single IP address that facilitates access to a network. Thus, a netinterface is not necessarily related to a single hardware network card. As popular operating systems employ virtual network interfaces for a number of different reasons such as OS virtualization and IP aliasing capabilities (the ability of a single hardware network interface to associate to more than 1 IP address), ITPSL follows the same approach. Consequently, one 'netinterfaceexists' statement should be issued for each IP address the system associates to.

```
<netinterfaceexists>
        <ipversion> 4 | 6 </ipversion>
        <intip> ip | (ip(range))  | AND/OR/XOR (ip1, ip2,…) | AND/OR/XOR (ip(range1),
        iprange2…) </intip>
        <netmask> netmask | AND/OR/XOR/NOT (netmask1, netmask2)</netmask>
        <intmac> ‘any’ | macaddr | AND/OR/XOR/NOT (macaddr1, macaddr2) </intmac>
</netinterfaceexists>
```

Figure 6.30: The 'netinterfaceexists' ITPSL statement

Each of the specified IP addresses can be given in a number of different forms including single IPs or a range of IPs. As an example, let’s specify a network interface that listens on IPv4 address 192.168.14.135 with a standard 24 bit (Class C) subnet mask and MAC address of 00:1A:A0:15:A8:7B, as shown in Figure 6.31 below.

```
<netinterfaceexists>
        <ipversion> 4 </ipversion>
        <intip> 192.168.14.135 </intip>
        <netmask> 255.255.255.0 </intip>
        <intmac> 00:1A:A0:15:A8:7B </intip>
</netinterfaceexists>
```

Figure 6.31: Expressing the existence of a network interface

On the other hand, if we do not know the exact IP address that the interface will have at runtime, we could specify a range of IP addresses by using the OR operator.

```
<netinterfaceexists>
        <ipversion> 4 </ipversion>
        <intip>OR(192.168.14(3-250),192.168.12(3-250)</intip>
        <netmask> OR(255.255.255.0 , 255.255.0.0) </intip>
        <intmac> any </intmac>
</netinterfaceexists>
```

Figure 6.32: Existence of a network interface with alternative IP addresses

ITPSL also needs a way to encode the state of a computer system routing table. The fact that the system is attached directly to networks of interest should be flagged and this is when we can use the ‘routeexists’ statement. A possible scenario where 'routeexists' would be useful is to examine whether a laptop has been used on non-corporate networks that are suspicious or insecure. This information could then be correlated with other activities (transmission or reception of sensitive files). In simple words, 'routexists' can act as a map of hinting where the device was located. Figure 6.33 displays the syntax of 'routeexists'.

```
<routeexists>
        <ipversion> 4 | 6 </ipversion>
        <netip> netip | AND/OR/XOR/NOT (netip1, netip2,...) </netip>
        <netmask> netmask | AND/OR/XOR/NOT (netmask1, netmask2)</netmask>
        <viainterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viainterface>
        [
        <gateway>gwip | AND/OR/XOR/NOT (gwip1,gwip2)</gateway>
        ]
</routeexists>
```

Figure 6.33: The 'routeexists' ITPSL statement

As an example of using 'routeexists', Figure 6.34 displays the routing table and the network interface of a real Linux workstation.

```
Destination        Gateway            Genmask          Flags   MSS    Window irtt Iface
129.240.235.0      0.0.0.0            255.255.255.0    U       0       0       0 eth0
192.168.122.0      0.0.0.0            255.255.255.0    U       0       0       0 virbr0
0.0.0.0            129.240.235.1      0.0.0.0          UG      0       0       0 eth0

eth0      Link encap:Ethernet  HWaddr 00:1A:A0:15:A3:7C
          inet addr:129.240.235.130  Bcast:129.240.235.255  Mask:255.255.255.0
virbr0    Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
```

Figure 6.34: Routing information from a Linux workstation

```
<subblock>
        <subop> AND </subop>
                <routeexists>
                        <ipversion> 4 </ipversion>
                        <netip> 129.240.235.0 </netip>
                        <netmask> 255.255.255.0 </netmask>
                        <viainterface>
                                <netinterfaceexists>
                                        <ip> 129.240.235.130 </ip>
                                </netinterfaceexists>
                        </viainterface>
                        <gateway> 129.240.235.1 </gateway>
                </routeexists>
                <routeexists>
                        <ipversion> 4 </ipversion>
                        <netip> 192.168.122.0 </netip>
                        <netmask> 255.255.255.0 </netmask>
                        <viainterface>
                                <netinterfaceexists>
                                        <ip> 192.168.122.1 </ip>
                                </netinterfaceexists>
                        <gateway> 129.240.235.1 </gateway>
                </routeexists>
</subblock>
```

Figure 6.35: An example of specifying routing configuration in ITPSL

Figure 6.35 displays part of the ITPSL markup that specifies the above networking configuration using one signature subblock with two 'routeexists' statements. The subop AND operator ensures that both routeexists statements must be true, in order to validate the contents of the subblock. Note how ‘netinterfaceexists’ statements are encapsulated inside a ‘routeexists’ statement, in order to bind the route to its native interface. This nesting mechanism is similar to the encapsulation of ‘filexists’ statement inside a ‘direxists’ one, as discussed in earlier sections. This is the reason the encapsulated ‘netinterfacexists’ statements lack the rest of the required tags (<ipversion>,<netmask>, etc) as these are taken in context from the relevant ‘routeexists’ tags.

6.6.2 Network access ability statements

```
<usercanaccessnet>
        <username> username </username>
        <ipversion> 4 | 6 </ipversion>
        <source> external | allnets | local | ip | FQDN | net[/subnet] | AND/OR/XOR/NOT(ip1,
ip2,    …) | AND/OR/XOR/NOT(FQDN1, FQDN2,…) | AND/OR/XOR/NOT(ip1,FQDN1,ip2,
…) |            AND/OR/XOR (net1[/subnet1],net2[/subnet2])</source>
        <destination> external | allnets | local  | ip | FQDN | net[/subnet] |
AND/OR/XOR/NOT(ip1,             ip2,…) | AND/OR/XOR/NOT(FQDN1, FQDN2,…) |
        AND/OR/XOR/NOT(ip1,FQDN1,ip2,…) | AND/OR/XOR
(net1[/subnet1],net2[/subnet2])     </destination>
        <singlenet> yes | no </singlenet>
        [
        <viasourceinterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viasourceinterface>
        <viadestinationinterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viadestinationinterface>
        <quantity> (over|less|equal) number </quantity>
        <sport> any | port |  (port1-portn) | AND/OR/XOR/NOT (port1, port2,…) </sport>
        <dport> any | port |  (port1-portn) | AND/OR/XOR/NOT (port1, port2,…) </dport>
        <transport> udp | tcp </transport>
        <URI> URI | AND/OR/XOR/NOT (URI1, URI2…)</URI>
        <viaroutepath> ip | FQDN | AND/OR/XOR/NOT (ip1,ip2,...) | AND/OR/XOR/NOT
                (FQDN1, FQDN2,…) |  AND/OR/XOR/NOT(ip1,FQDN1,ip2,…)
</viaroutepath>
        <receives> string | AND/OR/XOR/NOT (string1, string2, string3) </receives>
        ]
</usercanaccessnet>
```

Figure 6.36: The 'usercanaccessnet' ITPSL statement

As part of ITPSL's ability to express potential network access scenarios, 'usercanaccessnet' examines whether a network resource is reachable by a certain user specified by the <username> tag (Figure 6.36). The network resource is expressed by a number of mandatory and optional criteria. The source and destination IP addresses constitute the mandatory part, whereas source and destination ports could be optionally specified. The combination of IP addresses and ports for the source and destination addresses define the endpoints of interest. In ITPSL, an endpoint is the entity on one end of a transport layer connection (<transport> tag). As previous paragraphs discussed, the network statements are heavily based on the TCP/IP suite.

With the source and destination endpoints defined, the IP and transport layer details are taken care of. However, the language user can also specify additional criteria that might concern the application layer of the TCP/IP protocol. The <URI> tag enables one to express a network resource by using the Uniform Resource Identifier (URI) format, as described in RFC 3986 [138]. URI's are used to specify all sorts of resources not only network ones. However, their widespread use in network enabled protocols makes them a quite handy mechanism to specify additional details about network access issues.

The URI inside the tag follows the syntax of the URI scheme as defined in RFC 3986 [138]:

<scheme name> : <hierarchical part> [? <query>] [# <fragment>]

The 'scheme name' hints as to which application layer protocol is in use, the hierarchical part reveals details that should be consistent with other specified criteria such as the specified source and destination endpoints and then the rest of the parts give further information about the resource to access. For example, if one specifies the following URI as part of a ‘usercanaccessnet’ statement:

http://cnkeeper.uio.no:8080/mrs-web/status.do?method=databanks

the scheme name (http) informs that the transport connection carries info to the Hyper Text Transport Protocol, the 'cnkeeper.uio.no:8080’ scheme should be the same as the destination IP and port criteria and finally the ‘/mrs-web/status.do?method=databanks’ part submits a query of “databanks” in the 'status.do' method.

The <viasourceinterface> and <viadestinationinterface> tags allow the signature author to optionally choose specific network interfaces for outbound and inbound endpoint traffic in systems that have more than one network interface. They both embed ‘netinterfaceexists’ statements and the following paragraphs will discuss scenarios where they can be used.

The <viaroutepath> tag is another interesting optional criterion that checks to see whether the specified network resource is accessed via a certain route or routes. For example, a security analyst might like to check whether access to a certain network resource can be achieved via a certain network path as part of a penetration testing procedure. In that case, a list of the IP or FQDN addresses of some (or all) the intermediate routers could be entered.

The <singlenet> and <quantity> tags perform quantitative checks on whether we are specifying one or more networks and on the number of endpoints we can access respectively, as we did with various file statements.

Finally, the <receives> tag is an optional mechanism that enables the ‘usercanaccessnet’ statement to check more information about the destination endpoint. For example, when we connect to an FTP server, the process of initiating the connection includes the FTP server sending some info about the server, by means of the 220 FTP code [139] or by other codes. This is similar to other network servers and thus the feature could be used to identify a specific service, so that one can be more confident about connecting to the right endpoint.

The <receives> tag implies that the ‘usercanaccessnet’ statement will make the ITPSL engine launch a process that initiates the connection as specified by the endpoint criteria. This process will be launched with the credentials of the user, in order to replicate as closely as possible the conditions of the net access check. In addition, the ITPSL engine will look at the port number to infer what kind of service is being checked. Thus, the string specified by using the <receives> tag is taken in the context of the service being specified by the port number(s).

The fact that the signature author has a feature to check network accessibility by launching processes has a certain computational overhead for the system the check is performed, as well as for the destination system that offers the network service. Whilst the ITPSL engine will attempt to optimize the execution rate of this check, caution should be exercised by the signature author to specify checks in an efficient and proper manner, particularly when probing systems on the Internet.

In order to clarify the usage ‘usercanaccessnet’ statements, let’s examine a range of scenarios. If we wish to check whether user ‘johnc’ can access an ftp server that might be listening on a range of potential IP addresses by means of a single FTP connection and use the <receive> tag mechanism to check that we are talking to a specific FTP server. We could then write the ITPSL markup of Figure 6.37 below.

```
<usercanaccessnet>
        <username> johnc </username>
        <ipversion> 4 </ipversion>
        <source>testsystem.exampledomain.org</source>
        <destination>OR (129.240.235.45, 129.240.235.56)</destination>
        <singlenet> yes </singlenet>
        <dport> 21 </dport>
        <transport>tcp</transport>
        <receives> “DMVS FTP server” </receives>
</usercanaccessnet>
```

Figure 6.37: An example of 'usercanaccessnet' usage

'testsystem.exampledomain.org’ is the source IP. Specifying the source IP of the single local interface is sufficient, if the system has only one network interface. In the above example, we could have also specified the source IP as:

```
<source>local</source>
```

The word local could automatically be resolved to the IP of the local network interface automatically, provided that the system had only one network interface. However, if the origin system had more than one network interface (multi-homed), we could also be more specific and specify the source network interface by embedding a ‘netinterfaceexists’ statement. We demonstrate below this case and we now try to check whether the user can access simultaneously more than 3 FTP connections (Figure 6.38).

```
<usercanaccessnet>
        <username> johnc </username>
        <ipversion> 4 </ipversion>
        <source> local </source>
        <viasourceinterface>
                <netinterfaceexists>
                        <ipversion> 4 </ipversion>
                        <intip> 192.168.14.135 </intip>
                        <netmask> 255.255.255.0 </intip>
                        <intmac> 00:1A:A0:15:A8:7B </intip>
                </netinterfaceexists>
        </viasourceinterface>
        <destination> OR (129.240.235.45, 129.240.235.56)</destination>
        <singlenet> no </singlenet>
        <dport> 21 </dport>
        <transport>tcp</transport>
        <quantity>over 3 </quantity>
        <receives> “DMVS FTP server” </receives>
</usercanaccessnet>
```

Figure 6.38: Checking for the ability to initiate a number of FTP sessions

The above code instructs the ITPSL engine to launch 3 simultaneous sessions to the specified FTP server starting from a specific interface, specified as local and accompanied by a <viasourceinterface> tag.

The previous example and its variations examined outbound network access. ‘usercanaccessnet’ can also accommodate scenarios for inbound connections. A user can also access network resources when the user account receives traffic from remote endpoints. Thus, it should be possible to check for the ability to connect to a certain range of ports (in TCP/IP lingo we say we ‘bind’ a process to an endpoint) and prove that a particular user account could potentially receive traffic from a remote endpoint. The word “potentially” in the previous sentence is important. The ITPSL engine will not go ahead and launch a test connection from any remote endpoint. We have means to launch outbound network access checks to any reachable remote endpoint. On the other hand, initiating inbound network access from any reachable remote endpoint is not always possible. As a result, when specifying inbound net access checks, valid remote endpoints are considered only the endpoints of computer systems that are under the control of the ITPSL engine. The engine will first check on the local system to see if the user can bind to the specified local endpoints. After that, it will attempt to connect to that local endpoint from a valid remote endpoint. This consequence limits the scope of the network checks, however it is still important for two reasons:

- It provides useful host-level criteria for network processes (whether a user can bound a process to a network port is an important step for network access).
- Secondly, it provides a perspective of host inbound network connectivity from certain internal or external networks. This could be of importance to verify open gaps in network security and hence express network related threats.

Let's assume that we need to test whether 'johnc' is able to bind a server daemon to port 8080 and accept a connection from a range of internal networks. We assume that the host has a single network interface, so we write the ITPSL markup of Figure 6.39.

```
<usercanaccessnet>
        <username>johnc</username>
        <ipversion> 4 </ipversion>
        <source>AND(192.168.14.0, 192.168.45.0)</source>
        <destination>local</destination>
        <dport>8080</dport>
        <transport>tcp</transport>
        <singlenet>yes</singlenet>
</usercanaccessnet>
```

Figure 6.39: Checking port binding and connection acceptance ability

If the test should include external (beyond the organizational boundaries) endpoints as well as internal ones, we could re-write the source tag:

<source>AND(192.168.14.0, 192.168.45.0, external)</source>

If we wanted to test inbound connectivity from every possible remote network, we could use the 'allnets' keyword in the <source> tag:

<source>allnets</source>

This means that the ITPSL engine would initiate a single test connection from every possible internal and external network it controls, in an attempt to verify global connectivity.

If the host has more than one network interface, we need to express the interface to bind the process that will be the receiving end. Hence, we use the <viadestinationinterface> tag to encapsulate information about the destination interface (in contrast to outbound traffic checks, where we used the <viasourceinterface> tag). Figure 6.40 provides a suitable example.

```
<usercanaccessnet>
    <username> johnc </username>
    <ipversion> 4 </ipversion>
    <source>AND(192.168.14.0, 192.168.45.0)</source>
    <viadestinationinterface>
        <netinterfaceexists>
            <ipversion> 4 </ipversion>
            <intip> 192.168.14.135 </intip>
            <netmask> 255.255.255.0 </intip>
            <intmac> 00:1A:A0:15:A8:7B </intip>
        </netinterfaceexists>
    </viadestinationinterface>
    <dport>8080</dport>
    <transport>tcp</transport>
    <singlenet>yes</singlenet>
</usercanaccessnet>
```

Figure 6.40: Port binding and network connectivity test on a multi-home host

The ‘groupcanaccessnet’ functions similarly to a ‘usercanaccessnet’ statement, apart from the fact that the checks are performed against a number of users, in order to make easier to perform a specific check against many users. Figure 6.41 shows the syntax os the 'groupcanaccessnet' ITPSL statement.

```
<groupcanaccessnet>
        <groupid>groupname | (username1, username2)</groupid>
        <ipversion> 4 | 6 </ipversion>
        <source> external | allnets | local | ip | FQDN | net[/subnet] | AND/OR/XOR/NOT(ip1,
ip2,    …) | AND/OR/XOR/NOT(FQDN1, FQDN2,…) | AND/OR/XOR/NOT(ip1,FQDN1,ip2,
…) |    AND/OR/XOR (net1[/subnet1],net2[/subnet2])</source>
        <destination> external | allnets | local  | ip | FQDN | net[/subnet] |
AND/OR/XOR/NOT(ip1, ip2,…) | AND/OR/XOR/NOT(FQDN1, FQDN2,…) |
AND/OR/XOR/NOT(ip1,FQDN1,ip2, …) | AND/OR/XOR
(net1[/subnet1],net2[/subnet2]) </destination>
        <singlenet> yes | no </singlenet>
        [
        <quantity> (over|less|equal) number </quantity>
        <viasourceinterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viasourceinterface>
        <viadestinationinterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viadestinationinterface>

        <sport> any | port |  (port1-portn) | AND/OR/XOR/NOT (port1, port2,…) </sport>
        <dport> any | port |  (port1-portn) | AND/OR/XOR/NOT (port1, port2,…) </dport>
        <transport> udp | tcp </transport>
        <URI> URI-name </URI>
        <viaroutepath> ip | FQDN | AND/OR/XOR/NOT (ip1,ip2,...) | AND/OR/XOR/NOT
(FQD            N1, FQDN2,…) | AND/OR/XOR/NOT(ip1,FQDN1,ip2,…) </viaroutepath>
        <receives> string | AND/OR/XOR/NOT (string1, string2, string3) </receives>
        ]
</groupcanaccessnet>
```

Figure 6.41: The ITPSL 'groupcanaccessnet' statement

```
<groupcanaccessnet>
        <groupid>accounting</groupid>
        <ipversion> 4 </ipversion>
        <source>local</local>
        <destination> internalweb.chicago.pcl.com</destination
        <dport> 80 </dport>
        <transport> tcp </transport>
        <URI> http://internalweb.chicago.pcl.com/reports/authentication.html </URI>
        <viaroutepath> NOT (129.240.250.10) </viaroutepath>
        <receives> ”Please enter your password:” </receives>
</groupcanaccessnet>
```

Figure 6.42: Testing an unsafe network access scenario for a user group

As with the relevant file group statements, the <groupid> tag specifies either an OS file group or a list of usernames against which the checks will be performed.

For example, we can express an outbound network access check for the ‘accounting’ group users, in order to examine whether each of the users can connect to a particular web page as shown in Figure 6.42.

The example of Figure 6.42 includes the use of the <viaroutepath> tag. In this case, the tag makes the entire statement valid when the route path from the system the check is performed to the web server does not include the network 129.240.235.10. This could be of importance in a scenario where the signature author wishes to check whether access to the web page is facilitated in a secure way, as part of a threat assessment scenario. If the specified web page is an important authentication gateway, information is transmitted unencrypted (the URI specifies http and not https) and information flows to the web server not via a specified VPN route, one could assume that this is an unsafe scenario and thus could use this statement as part of expressing an unsafe net access scenario.

6.6.3 Network access statements

We have seen so far ITPSL network statements that express the existence of routes and interfaces, statements that help the signature author express network access checks against one or more users. ‘netaccess’ expresses signs of current and past network activity by one or more users, as part of misuse threat expression scenarios, as fileaccess and diraccess do for file operations. Resembling the philosophy of the similarly relevant file access statements, ‘netaccess’ uses the <pattern> tag to express relevant network activity by using two types of access patterns:

- Net access on a temporal basis: Specifies whether network access activity is current (time of the signature check) or when access takes place between certain time periods of the present day, the day before, the present week or the present month.
- Net access on a temporal frequency basis: When it is important to specify how often network access activity takes place within a specified period of time.

The other important aspect of ‘netaccess’ is that it provides a mechanism that relates network activity to user(s) (<byuser> and <bygroup> tags). ‘usercanaccessnet’ and ‘groupcanaccessnet’ launch sessions to test network accessibility by using the user/group credentials under controlled circumstances. ‘netaccess’ will have to link existing or past activity to users. This is an important feature in the process of estimating threats, as suspicious network activity must be accounted to users. Figure 6.43 displays the syntax of the 'netaccess' statement.

```
<netaccess>
        <ipversion> 4 | 6 </ipversion>
        <source> local | any | ip| FQDN | net[/subnet] | AND/OR/XOR/NOT(ip1, ip2,…) |
        AND/OR/XOR/NOT(FQDN1, FQDN2,…) | AND/OR/XOR/NOT(ip1,FQDN1,ip2,…) |
        AND/OR/XOR (net1[/subnet1],net2[/subnet2]) </source>
        <destination> local | any | ip | FQDN | net[/subnet] | AND/OR/XOR/NOT(ip1, ip2,…) |
        AND/OR/XOR/NOT(FQDN1, FQDN2,…) | AND/OR/XOR/NOT(ip1,FQDN1,ip2,…) |
        AND/OR/XOR (net1[/subnet1],net2[/subnet2]) </destination>
        <singlenet> yes | no </singlenet>
        [
        <quantity> (over|less|equal) number </quantity>
        <viasourceinterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viasourceinterface>
        <viadestinationinterface>
                <netinterfaceexists>…</netinterfaceexists>
        </viadestinationinterface>
        <sport> any | port |  (port1-portn) | AND/OR/XOR/NOT (port1, port2,…) </sport>
        <dport> any | port |  (port1-portn) | AND/OR/XOR/NOT (port1, port2,…) </dport>
        <transport> udp | tcp </transport>
        <URI> URI-name </URI>
        <pattern> AND/OR/XOR/NOT ( now| hh-hh today | hh-hh (x | (0-30) ) days ago
[(more-         than|less-than) x times | every (minute | hour | (Sunday-Saturday))  ) </pattern>
        <viaroutepath> ip | FQDN | AND/OR/XOR/NOT (ip1,ip2,...) | AND/OR/XOR/NOT
(FQD  N1, FQDN2,…) |  AND/OR/XOR/NOT(ip1,FQDN1,ip2,…) </viaroutepath>
        <byapplication> program-name | path | AND/OR/XOR/NOT (path1,path2) </viaappli-
cation>
        <byuser> username </username>
        <bygroup> groupname | (username1,username2)</bygroup>
        ]
</netaccess>
```

Figure 6.43: The ITPSL 'netaccess' statement

Most operating systems provide utilities that monitor network activity at various levels. However, this kind of monitoring does not usually include tools that link network access to users. A good example of such a utility is netstat [140] found on both Windows and Unix/Linux operating systems. ‘netstat’ lists remote and local connection endpoints, the state of the connection and other useful information but it does not associate connection endpoints to users. Other utilities such as ‘lsof’ [94] found in the Unix/Linux family of operating systems can associate connections to users.

It is also possible to be more precise and associate endpoint, user and application information. The <viaapplication> tag complements the association criteria, so that one can express “this network endpoint accessed by this user via that application” in a netaccess statement.

```
<netaccess>
        <ipversion> 4 | 6 </ipversion>
        <source> local </local>
        <destination> www.helpfulltorrents.com </destination>
        <singlenet> yes </singlenet>
        <transport> tcp </transport>
        <URI> http://www.helpfulltorrents.com/torrentsearch </URI>
<pattern> AND (  20-24 1 days ago, more-than 2 times ) </pattern>
<viaapplication> OR (/usr/bin/firefox, /usr/local/bin/azureus, firefox, azureus) </viaapplication>
<byuser> johna </username>
</netaccess>
```

Figure 6.44: A 'netaccess' usage example

The rest of the netaccess statement tags are identical to the statements of the previous section and thus, we begin describing potential usage examples in the following paragraphs. If we wish to examine whether the web page ‘http://www.helpfulltorrents.com/torrentsearch’ has been accessed more than two times, between 20 and 24 hours yesterday by user johna, and by using either the ‘firefox’ or the ‘azureus’ applications, we could express that with the ITPSL markup of Figure 6.44.

The <byuser> tag relates the intercepted endpoint activity to a specific user. In addition, the <viaapplication> tag will extend the relation to an executed application by the user. Thus, the above statement relates the endpoint access to the user as well as to the application the user executes to create the described endpoint.

```
<netaccess>
        <ipversion> 4 </ipversion>
        <source> NOT (130.135.201/24) </source>
        <destination> corp.smtp.abc.com </destination>
        <singlenet> yes </singlenet>
        <transport> tcp </transport>
        <dport> 25 </transport>
        <pattern> 9-18 (0-30) days ago </pattern>
        <viaroutepath> NOT scarlet.vpnaccess.abc.com </viaroutepath>
</netaccess>
```

Figure 6.45: Checking whether a user utilized a non trusted network

Another example checks whether a group of users has been accessing a corporate SMTP server via a routepath that does not involve the corporate VPN during working hours within the last month. In a threat scenario, routing data via a non-trusted network could indicate an important data compromise activity. Figure 6.45 displays the markup that expresses this condition.

6.7 ITPSL process execution statements

Users interact with the computer system by means of executing programs. A process is “an instance of a program in execution” [42], which of course induces file and network level activity that the two previous types of ITPSL statements are able to express. However, in order to be able to fully express misuse scenarios, one needs to focus also on how programs are executed, not only on the effects of programs at file and network level. The process execution statements give the signature author means to express aspects of misuse threats that relate to process execution events.

There are 4 types of process execution statements:

- **General process execution description (procexec)**: A single statement ‘procexec’ describes the execution of a process and does not relate the process to a user. Sometimes it might be worth describing a process regardless of its user credentials and later paragraphs will discuss such cases. A ‘procexec’ can be embedded in other types of process description statements.
- User related process execution (userexec, groupexec): The same as ‘procexec’ but they associate the described process with a certain user (userexec) or user group (groupexec).
- **In sequence user related process execution (userexecsequence, groupexecsequence)**: Earlier paragraphs discussed the ability to express temporal information in relation to file and network statements. Every threat materializes as a sequence of related steps. Whilst it is possible to use other language features (the ‘AND’ logical operator, the ‘as_a_result_of’) to express a sequence of events, process execution is a good target for providing a mechanism to explicitly describe an ordered sequence of actions relevant to a misuse threat scenario. This is not only because processes are the focal high-level point that orchestrates file and network level events. A range of research techniques have focused on command execution [141] thus making the ability to encode a process execution sequence in ITPSL useful. As a result, ‘userexecsequence’ relates a process execution to a user, whereas ‘groupexecsequence’ performs the same for groups.
- **Hardware operation statements (hardwareop)**: Certain misuse threats might be associated with specific hardware level changes. A good example is information theft by means of mobile media. Hence, the ITPSL vocabulary should include statements that are able to detect the removal or addition of certain devices. As this sort of operation is process based (a

program performs the hardware changes), the most reasonable approach involves placing ‘hardwareop’ under the process execution statements.

All types of execution statements provide the ability to describe the temporal basis of process execution events. In contrast to the file and network ITPSL statements where the signature author had the ability to check file and network access scenarios, execution statements do not provide a mechanism to test whether a user/group can execute certain processes for two reasons. Firstly, executing a program of unknown origin might make the system vulnerable to malicious code. Secondly, safer mechanisms that can facilitate whether a user/group can execute a program exist. The ‘usercanaccessfile’ and ‘groupcanaccessfile’ statements can check by means of the <accessrights> tag for execution rights and thus deduce whether a user could potentially execute a program without actually running it.

6.7.1 The general process execution statement

The ‘procexec’ statement (Figure 6.46) is the workhorse of process execution statements and allows us to describe the fact that one or more instances of a program (<singleprocess> and <quantity> tags) are being executed or have been executed in the past as described by the <pattern> tag. The <name> and <path> tags describe the exact location (or a combination of names and locations), whereas the <argumentlist> tag specifies also any arguments or options passed to the program.

```
<procexec>
        <name> process-name | AND/OR/XOR/NOT (process-name1, process-name2)</name>
        <path> any | path | OR/XOR/NOT(path1, path2) </path>
        <singleprocess> yes | no </singleprocess>get
        <quantity> (over|less|equal) number </quantity>
        <argumentlist>(argument-list)   |   AND/OR/XOR/NOT(   (argument-list1),   (argu-
ment-list2)    …) </argumentlist>
        <pattern> AND/OR/XOR/NOT ( fromnow| hh-hh today | hh-hh (x | (0-30) ) days ago
[(more-        than|less-than) x times | every (minute | hour | (Sunday-Saturday))  ) </pattern>
        ]
</procexec>
```

Figure 6.46: The 'procexec' ITPSL statement

For instance, if one wishes to specify that a MySQL server process was running between 7 and 23 hours of the previous day, we could write the following ‘procexec’ statement, as shown in Figure 6.47.

```
<procexec>
        <name> mysqld </name>
        <path> OR(/usr/bin/, /usr/local/bin) </path>
        <singleprocess> yes </singleprocess>
        <argumentlist> (/etc/my.cnf, 3306) </argumentlist>
        <pattern> 07-23 1 days ago </pattern>
</procexec>
```

Figure 6.47: Specifying the execution of a process with 'procexec'

The <path> tag dictates the two possible locations 'mysqld' might be running from (the first one that matches will be considered as a hit for the signature). In addition, the argument list focuses on desired operational parameters for the MySQL server that concern the location of its main configuration file and the port it is listening on. The argument list of the above example is generic and it involves only the actual arguments. These arguments may be expressed in a more specific way and include, for example, the actual command-line switches, as shown below:

<argumentlist> (--defaults-file=/etc/my.cnf, --port=3306) </argumentlist>

The generic argument list form acts as a regular expression matching only the relevant parts of the arguments and could be used when command-line argument switches are likely to change, whereas the second specific form could be used to perhaps match the operation against a certain MySQL version, for which the signature author knows the exact form of the command-line switches.

A ‘procexec’ statement does not relate the process execution to a user but it just describes the process execution itself. This is true for a standalone ‘procexec’ statement. However, when ‘procexec’ is encapsulated inside another execution statement, a user relation exists taken by context. The following section will discuss this issue.

6.7.2 User related process execution statements

When the execution should be connected to a user entity, a ‘userexec’ statement should be used instead (Figure 6.48). ‘userexec’ has the same tags as a ‘procexec’ statetement, with the exception of <username> in order to identify the user in question.

The reason for having two separate types of statements (general and user related process execution) is to be able to combine the ability to express conditions that are relevant to a user and non user related process events with semantic clarity. This could also add accuracy to the description of a process execution event. For instance, in the previous example where we described the execution of a MySQL server process, we could also include the execution of a MySQL client process by a user as shown in the ITPSL subblock of Figure 6.49.

```
<userexec>
        <username> username </username>
        <name>  process-name  |  AND/OR/XOR/NOT  (process-name1,  process-name2)
</name>
        <path> any | path | OR/XOR/NOT(path1, path2) </path>
        <singleprocess> yes | no </singleprocess>
        [
        <quantity> (over|less|equal) number </quantity>
        <argumentlist>(argument-list) | AND/OR/XOR/NOT( (argument-list1), (argumen
        t-list2)…) </argumentlist>
        <pattern> AND/OR/XOR/NOT ( fromnow| any |  hh-hh today | hh-hh (x | (0-30) ) days
        ago [(more-than|less-than) x times | every (minute | hour | (Sunday-Saturday))  ) </pat-
tern>
        ]
</userexec>
```

Figure 6.48: The 'userexec' ITPSL statement

```
<subblock>
        <subop> AND </subop>
        <procexec>
                <name> mysqld </name>
                <path> OR(/usr/bin/, /usr/local/bin) </path>
                <singleprocess> yes </singleprocess>
                <argumentlist> (/etc/my.cnf, 3306) </argumentlist>
                <pattern> 07-23 1 days ago </pattern>
        </procexec>
        <userexec>
                <username>johna</username>
                <name> mysql </name>
                <path> OR(/usr/bin/, /usr/local/bin) </path>
                <singleprocess> yes </singleprocess>
                <argumentlist> OR((-ujohna, -hlocalhost), (johna, localhost)</argumentlist>
                <pattern>07-23 1 days ago </pattern>
        </userexec>
</subblock>
```

Figure 6.49: User attributable process execution versus non user attributable process execution

By combining the two statements with an AND subop operator as shown above, we describe the process environment in a more specific way. The ITPSL engine will check for the existence of a MySQL client process ran by user ‘johna’ that connects to a local MySQL server process. The execution of a local MySQL server is described by the accompanying ‘procexec’ and could ran with the credentials of any user. A secure MySQL server configuration dictates that the MySQL server process does not map to a specific user entity. This means that the userid of the server process might not be usable by actual users. Thus, if the user had tried to run the client connecting to a local MySQL server without the MySQL server process running on the machine and a ‘procexec’ had not been specified, the signature would match the fact that a MySQL client had run. However, if the intention was to describe a complete database access scenario in this case, matching the client only without checking for a MySQL server process would be pointless and inaccurate.

A ‘groupexec’ statement functions similarly to a ‘userexec’ statement but it checks the process execution specification criteria against a group of users. Similarly to the file and network ITPSL statements, the word group refers to an OS file group or a list of usernames that form a logical group. Figure 6.50 shows the syntax of the 'groupexec' statement.

```
<groupexec>
        <groupname> groupname | AND/OR/XOR/NOT(username1, username2) </username>
        <name> process-name | AND/OR/XOR/NOT (process-name1, process-name2)
</name>
        <path> any | path | OR/XOR/NOT(path1, path2) </path>
        <singleprocess> yes | no </singleprocess>
        [
        <quantity> (over|less|equal) number </quantity>
        <argumentlist>(argument-list) | AND/OR/XOR/NOT( (argument-list1), (argu-
ment-list2) …) </argumentlist>
        <pattern> AND/OR/XOR/NOT ( now| hh-hh today | hh-hh (x | (0-30) ) days ago
[(more-    than|less-than) x times | every (minute | hour | (Sunday-Saturday)) ) </pattern>
        ]
</groupexec>
```

Figure 6.50: The 'groupexec' ITPSL statement

```
<groupexec>
        <groupname> engineering </groupname>
        <name> thunderbird-bin </name>
        <path> /usr/bin </path>
        <singleprocess> no </singleprocess>
        <quantity> over 3 </quantity>
        <pattern> now </pattern>
</groupexec>
```

Figure 6.51: A 'groupexec' usage example

For example, if one wants to check whether the ‘engineering’ file group is presently executing three or more thunderbird mail browsers, this could be expressed by a ‘groupexec’ statement as shown in Figure 6.51.

6.7.3 In sequence user related process execution statements

At other times, it might be necessary to express a sequence of process execution events related to a user as a pre-condition of a misuse threat scenario. This is a job for the ‘userexecsequence’ statement. Its syntax is displayed in Figure 6.52. Between the <procsequence> tags we can encapsulate a series of ‘procexec’ statements to encode an ordered collection of processes. This means that the ITPSL engine will try to match the process execution events in the order they are listed. It also means that the ‘procexec’ statements are implicitly related to the user defined by the <username> tag. This last bit is confusing, but as earlier paragraphs warned, when a ‘procexec’ statement is encapsulated inside another execution statement, a user relation is implied in context to the encapsulating statement.

```
<userexecsequence>
    <username> username | OR/XOR/NOT (username1,username2) </username>
    <timewindow> from-now | within x minute/hour/days | hh-hh today | hh-hh x days ago
    </timewindow>
    <procsequence>
        <procexec>…</procexec>
        <procexec>…</procexec>
        [
        <oneof>
            (<procexec>…</procexec>)
            (<procexec1>…</procexec1> <procexec2>…</procexec2>)
        </oneof>
        ]
        …
    </procsequence>
    [
    <validmatch>  x out-of t processes | threshold(w)=(scorea,scoreb,scorec…scoret)
    </validmatch>
    ]
</userexecsequence>
```

Figure 6.52: The 'userexecsequence' ITPSL statement

The optional <oneof> block inside the <procsequence> tag provides a way to make the process sequence polymorphic. If a particular step consists of many possible command combinations, they can all be specified inside the <one of> block and the one the first one that matches will be picked up by the ITPSL engine.

Another important issue with ‘userexecsequence’ is that of specifying the time window to try and match the encoded process execution sequence. The various steps of the specified program execution sequence might be separated by seconds, minutes, hours or even days. As a result, the signature author must instruct the ITPSL engine how far back it should go to look for the first command and/or when to end the search. Previous execution statements used the <pattern> tag in order to specify single execution events. However, specifying temporal information in relation to a sequence of events requires different semantics. The <timewindow> tag provides these semantics and later paragraphs will explain the concept in more detail.

Finally, the optional <validmatch> tag provides an additional refinement mechanism when it comes to the validity of the specified execution sequence. Ideally, all of the specified processes should match in order for the ‘userexecsequence’ statement to be valid. This is the default behavior of the ITPSL engine, when a <validmatch> tag is not specified. Caution should be exercised by the signature author, in order not to include processes that are irrelevant to the threat specification or miss execution events that could be important. In other words, a process execution sequence when expressed as a threat pre-condition, is as good as the ability of the signature author to isolate and express the relevant events. This of course stands true for every part of the ITPSL signature.

However, to accommodate for uncertainty created by the variety of commands that could be employed in sequence to create a threat, <validmatch> allows the signature author to specify a validity threshold in two possible ways:

- As a match score (x out of t processes, with t being the total number of specified processes in the statement) OR
- As a total threshold score (w) accompanied by a scored process weight list, so that some of the processes are considered more representative as threat indicators than others. Note that w is always smaller that the total score from the weight list on the right hand side (if w was equal to the total score of the weighted list, then all processes should be taken into consideration and then we would have the default case and no need to specify a <validmatch> tag).

threshold(w)=(scorea,scoreb,scorec…scoret)
where w<(scorea,scoreb,scorec…scoret)

Signature authors can think of <validmatch> as a kind of internal Weight Matrix that applies to process execution sequences. This last mechanism together with the <oneof> tag block gives more flexibility to the author to increase the specificity of a description by means of including more statements with varied degrees of relevance. It could also be used as a nice way to refine older signatures (keep the same process statements but change their weight and hence the scope of a threat specification).

In order to illustrate all these concepts, let’s assume that we are constructing a misuse signature oriented towards detecting the installation of P2P software. Hence, we wish to describe an execution sequence that includes the launching of a web browser, the decompression of a compressed archive and finally the running of the uncompressed executable by user ‘chrisc’ within the last 24 hours from the point the signature is checked. Figure 6.53 displays displays a suitably crafted 'userexecsequence' statement that encodes these conditions.

Note how the <pattern> tag is omitted from the encapsulated ‘procexec’ statements, as <timewindow> provides a time reference for the execution sequence search. However, the scope of the above process sequence example is not wide enough to accommodate a range of variations in the user actions. For example, what if the user had employed a different tool to decompress the downloaded executable and/or a different application to acquire it? We can widen the scope of the specification and re-write the relevant ‘procexec’ statements using logical operators as shown below by Figure 6.54. The XOR operator inside the relevant statement tags (<name>,<path> and <argumentlist>) includes more options for specifying user related processes.

```
<userexecsequence>
        <username> chrisc </username>
        <timewindow> within 1  days </timewindow>
        <procsequence>
                <procexec>
                        <name> firefox </name>
                        <path> OR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                </procexec>
                <procexec>
                        <name> tar</name>
                        <path> OR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                        <argumentlist> (xvfz, azureus*.tar.gz) </argumentlist>
                </procexec>
                <procexec>
                        <name> OR(azureus, azureus-bin) </name>
                        <path> any </path>
                        <singleprocess> yes </singleprocess>
                </procexec>
        </procsequence>
</userexecsequence>
```

Figure 6.53: P2P software installation detection by using userexecsequence

```
…
                 <procexec>
                        <name> XOR (mozilla, mozilla-bin, firefox, firefox-bin,ftp) </name>
                        <path> XOR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                </procexec>
                <procexec>
                        <name> XOR(tar, gzip)</name>
                        <path> XOR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                        <argumentlist> XOR ((xvfz, azureus*.tar.gz), (-d, azureus*.tar.gz))
                                </argumentlist>
                </procexec>
                …
```

Figure 6.54: Making the procexec statements polymorphic

At this point, we could also make use of the <validmatch> tag to weight the significance of the previously described processes. As a result, the entire statement could be re-written, as shown in Figure 6.55.

```
<userexecsequence>
        <username> chrisc </username>
        <timewindow> within 1  days </timewindow>
        <procsequence>
                <procexec>
                        <name> XOR (mozilla, mozilla-bin, firefox, firefox-bin,ftp) </name>
                        <path> XOR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                </procexec>
                <procexec>
                        <name> XOR(tar, gzip)</name>
                        <path> XOR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                        <argumentlist> XOR ((xvfz, azureus*.tar.gz), (-d, azureus*.tar.gz))
                                </argumentlist>
                </procexec>
                <procexec>
                        <name> OR(azureus, azureus-bin) </name>
                        <path> any </path>
                        <singleprocess> yes </singleprocess>
                </procexec>

        </procsequence>
        <validmatch> threshold(60)=(10,30,50)</validmatch>
</userexecsequence>
```

Figure 6.55: Using the 'validmatch' tag to refine a procexec statement

We use the <validmatch> tag to specify that the most important process execution event is the last one. It is normal to assign greater weighting to commands towards the end of the sequence. If a threat can be realized as a sequence of steps, then the further we get in the sequence, the closer we get towards the threat misuse scenario. The above example process sequence contains three process execution events. By assigning a threshold score of 60, and allocating the scores for the 3 commands as shown below, we essentially exclude the validation of the entire process sequence if the last step is not executed. This makes sense as the first process description (the launch of a browser) has no arguments and hence it carries little information (the user might have launched a browser to do something else) as no arguments were encoded. The second step is more relevant as now a certain file is being decompressed as dictated by the argument list. Finally, the last step is the concluding one that adds the most certainty and thus it gets the most points.

Still, the above re-write is far from perfect, as the above sequence of actions is incomplete. If the downloaded azureus executable is a tarball and the user employs the gzip utility as described, the file will still have to be ‘untared’. Thus, the sequence firefox->gzip->azureus is not right and we need a better way to express such a scenario. Figure 6.56 shows the way to do this by re-writing the <procsequence> block to include a <oneof> tag, in order to complete the sequence of commands by adding a little bit of polymorphism in the execution sequence.

The role of the <oneof> tag is to split the described sequence flow in two possible paths illustrating how execution sequence polymorphism could be achieved. One path encodes the user employing the tar command to decompress and untar the downloaded file in one step. The second alternative specifies two separate commands (gzip and tar) to perform the same task. Each of the possible candidates for what is essentially the second step of the process sequence is contained in a pair of parentheses inside the <oneof> block.

```
<userexecsequence>
        <username> chrisc </username>
        <timewindow> within 1  days </timewindow>
         <procsequence>
                <procexec>
                        <name> XOR (mozilla, mozilla-bin, firefox, firefox-bin,ftp) </name>
                        <path> OR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                </procexec>
                <oneof>
                        <procexec>
                                <name> tar</name>
                                <path> OR(/usr/bin/, /usr/local/bin) </path>
                                <singleprocess> yes </singleprocess>
                                <argumentlist> (xvfz, azureus*.tar.gz) </argumentlist>
                        </procexec>
                        <procexec>
                                <name>OR (gzip, gunzip)</name>
                                <path> OR(/usr/bin/, /usr/local/bin) </path>
                                <singleprocess>yes</singleprocess>
                                <argumentlist> (-d, azureus*.tar.gz) </argumentlist>
                        </procexec>
                        <procexec>
                                <name> tar </name>
                                <path> OR(/usr/bin/, /usr/local/bin) </path>
                                <singleprocess>yes</singleprocess>
                                <argumentlist> (xvf, azureus*.tar.gz) </argumentlist>
                        </procexec>
                </oneof>
                <procexec>
                        <name> OR(azureus, azureus-bin) </name>
                        <path> any </path>
                        <singleprocess> yes </singleprocess>
                </procexec>
        <validmatch> threshold(60)=(10,30,50)</validmatch>
</procsequence>
</userexecsequence>
```

Figure 6.56: The use of the 'oneof' tag in a 'userexecsequence' statement

An important detail relating to the threshold score when process description polymorphism is employed is that the <validmatch> tag evaluates only the successful match of a <oneof> tag as a single step. In the above example, the <validmatch> tag contained:

<validmatch> threshold(60)=(10,30,50)</validmatch>

Thus, if the signature matches a user executing either a tar or a combination of gzip and tar, only one step will be scored, even if the second <oneof> option contains two commands (gzip and tar). This is necessary to maintain the scoring consistency and also logically group alternative process execution steps (the important thing here is to encode the decompression and untar process of a file, not only to describe the actual commands).

All of the previous examples were based on a time window sequence search of 1 day:

<timewindow> within 1 days </timewindow>

Other combinations are of course possible as described by the statement syntax. One of them is peculiar. If one chooses the value ‘from-now’:

<timewindow> from-now </timewindow>

it will set the execution sequence search start reference point at the time of the signature check. The ITPSL engine will then try and collect the information, as the relevant commands gradually reach the logs. This can impact the signature execution time. If the <timewindow> tag specifies a period of search in the past, the data are already in the ITPSL engine log repository and hence the check will yield a result quickly. However, if the start search reference is set at the time of the signature check, the ‘userexecsequence’ statement will complete either by the time a matching sequence is found or after 30 days from the moment the search has started. The signature author should be aware of this detail for two reasons:

- From the perspective of execution timing, if ‘from-now’ is used as a <timewindow> value and the ITPSL engine waits for the detection of a matching process execution sequence, the time delay in the signature check will affect the entire signature (including the accompanying network and file statements), not only the execution part.

- In addition, the widespread use of ‘from-now’ will tie up computational cycles in the ITPSL engine (the engine has an active signature check for potentially long periods of time).

For these two reasons, a ‘from-now’ value should be used with caution only when it is necessary, for instance, in cases when it is known that a certain sequence might not have appeared in the past and is likely to appear in the future.

```
<groupexecsequence>
        <groupname> groupname | OR/XOR/NOT (username1,username2) </groupname>
        <timewindow> from-now | within x minute/hour/days | hh-hh today | hh-hh x days ago
        </timewindow>
        <procsequence>
                <procexec>…</procexec>
                <procexec>…</procexec>
                [
                <oneof>
                        (<procexec>…</procexec>)
                        (<procexec1>…</procexec1> <procexec2>…</procexec2>)
                </oneof>
                ]
                …
        </procsequence>
        [
        <validmatch>  x out-of t processes | threshold(w)=(scorea,scoreb,scorec…scoret)
        </validmatch>
        ]
</groupexecsequence>
```

Figure 6.57: The 'groupexecsequence' ITPSL statement

The ‘groupexecsequence’ statement functions in the same way as the ‘userexecsequence’ does, but it relates a process sequence to a group of users. For the purposes of clarity, Figure 6.58 displays a complete ‘groupexecsequence’ statement tailored to the previous scenario (P2P client installation) concerning the user group ‘sales’.

```
<groupexecsequence>
        <groupname> sales </groupname>
        <timewindow> within 1  days </timewindow>
        <procexec>
                <name> XOR (mozilla, mozilla-bin, firefox, firefox-bin,ftp) </name>
                <path> OR(/usr/bin/, /usr/local/bin) </path>
                <singleprocess> yes </singleprocess>
        </procexec>
        <oneof>
                <procexec>
                        <name> tar</name>
                        <path> OR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess> yes </singleprocess>
                        <argumentlist> (xvfz, azureus*.tar.gz) </argumentlist>
                </procexec>
                <procexec>
                        <name>OR (gzip, gunzip)</name>
                        <path> OR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess>yes</singleprocess>
                        <argumentlist> (-d, azureus*.tar.gz) </argumentlist>
                </procexec>
                <procexec>
                        <name> tar </name>
                        <path> OR(/usr/bin/, /usr/local/bin) </path>
                        <singleprocess>yes</singleprocess>
                        <argumentlist> (xvf, azureus*.tar.gz) </argumentlist>
                </procexec>
        </oneof>
        <procexec>
                <name> OR(azureus, azureus-bin) </name>
                <path> any </path>
                <singleprocess> yes </singleprocess>
        </procexec>
   </procsequence>
<validmatch> threshold(60)=(10,30,50) </validmatch>
</groupexecsequence>
```

Figure 6.58: A 'groupexecsequence' usage example

6.7.4 The hardware operation process statement

A ‘hardwareop’ statement describes the removal or addition of hardware components in a computer system (Figure 6.59). This could be useful when describing certain types of threats such as information theft scenarios where some sort of mobile medium is involved. The <devicetype> tag provides some way to abstract the type of device that was added or removed (<operation> tag). A ‘mobilestorage’ <devicetype> value is normally associated with USB/Firewire based media, whereas ‘storage’ refers to onboard devices. The device in question could also be optionally identified by a <deviceidstring> tag. The optional <storagepath> tag describes one or more potential mount points (or logical drive letters) where the device is expected to be attached.

```
<hardwareop>
        <operation>device-addition | device-removal </operation>
        <bus> usb | pci | any </bus>
        [
        <deviceidstring> 'device id string' | AND/OR/XOR/NOT ('devicestring1',
        'devicestring2',..</deviceidstring>
        <pattern> from-now | hh-hh today | hh-hh (x | (0-999) ) days ago </pattern>
        <userwasloggedon> username </userwasloggedon>
        <groupwasloggedon>groupname    |    AND/OR/XOR/NOT(username1,username2…)
        </groupwasloggedon>
        ]
</hardwareop>
```

Figure 6.59: The 'hardwareop' ITPSL statement

The <userwaslogged> and <groupwaslogged> tags help the signature author to maintain accountability by relating the hardware operation to a user or group of users respectively. The problem with most widely employed Operating Systems is that although they log device addition/removal events, they do not explicitly connect this kind of hardware operations to users. Hence, the user (or users)that was logged on at the time the hardware event took place is a good indication to include in a ‘hardwareop’ statement.

As an example of a ‘hardwareop’ statement consider the markup of Figure 6.60. We try to detect the hardware addition of a USB based mobile media device of a known model within the last 5 days between working hours and relate that event to user ‘chrisc’. Obviously the fact that user ‘chrisc’ was logged on might not be conclusive in terms of whether he has inserted the device (other users might also be logged in and the statement does not account for them). However, this statement should form part of a signature. Other parts of the signature can combine other checks (file activity on the specified mount point that would help determine whether ‘chrisc’ has inserted the device and is on a path towards an insider threat.

```
<hardwareop>
        <operation>device-addition</operation>
        <bus>usb</bus>
        <deviceidstring> OR ('MuVo-X', 'MuVo NX', ) </deviceidstring>
        <pattern> 08-17 6 days ago </pattern>
        <userwasloggedon> chrisc </userwasloggedon>
</hardwareop>
```

Figure 6.60: A 'hardwareop' statement example

6.8 Conclusions

This Chapter presented the ITPSL semantics and examples of their usage in simple scenarios. ITPSL should be implemented as an external DSL, in order to gain the flexibility of designing a language from scratch, without the dependencies of generic host languages (internal DSL approach). The translational semantic approach was used to create a framework where statements can cross-reference LUARM audit data (Figure 6.4). This framework is based on XML, a universal mechanism to describe data. XML's declarative syntax and statement hierarchy, as well as its standard mechanisms to transform (XSLT) and validate (XML schema) the structure of semantics provide a solid ground to build an insider threat specification language.

The ITPSL header (Figure 6.9) facilitates threat signature repository creation and maintenance with its ontology section. Furthermore, the Weight Matrix concept (Figure 6.8) provides a mechanism for the language to express decision theoretic information by means of associating certain statements (ITPSL sub-blocks) with a confidence weight. Hence, the signature author can craft threat prediction signatures based on his knowledge of a particular type of threat and classify certain events that are clear markers of forthcoming misuse incidents.

The main ITPSL body (Figure 6.10) contains file, network, process and hardware operation statements that describe the misuse act in threat detection or prediction context. The four different types of statements reflect the four different types of events monitored by the LUARM audit records.

The next Chapter discusses the implementation of the ITPSL compiler and evaluates the produced semantics in a series of scenarios.

Chapter 7 Realizing and evaluating the ITPSL

The previous chapter presented the ITPSL semantics, discussing the language directives and providing simple examples of their usage context. This chapter puts ITPSL and LUARM in action by evaluating the capabilities of the language against a range of carefully crafted IT misuse scenarios. The process of evaluating the capabilities of a Domain Specific Language (DSL) is easier than that of a generic programming language. Section 6.1 explained that a DSL is tailored to a specific domain rather than the generic range of tasks of a high level programming language [105]. However, benchmarking a DSL is still a process that requires a certain methodology and thus, the chapter starts by exploring this particular topic.

The discussion then moves on to present the ITPSL compiler prototype, the tool that converts ITPSL signatures to system level audit checks, in order to detect or predict the specified IT misuse scenarios. Finally, the chapter discusses the results of benchmarking ITPSL.

7.1 Evaluating the ITPSL effectiveness

Section 1.1 stated that the fifth and final objective of the thesis was to: "Make a prototype system and evaluate the proposed language against a range of hypothetical and real insider IT misuse scenarios.". The prototype system is the subject of the next section. This section raises the question of what is a meaningful evaluation for a threat specification language.

ITPSL is a Domain Specific Language (DSL) and thus its limited context (insider threat prediction and specification) should make it easy to produce a range of checks that benchmark the effectiveness of the language semantics. However, DSL construction is a peculiar process [142] as it requires both language and domain-specific expert knowledge, a combination which is hard to find.

The diversity of the insider IT misuse scenarios discussed in the third chapter combined with the number of functional requirements of section 4.3 indicate that the process of evaluating the effectiveness of ITPSL can be a non-trivial task.

At the beginning, DSLs like general purpose programming languages used to be evaluated empirically. They would be released to a user audience and evaluation would be performed by means of a long term process loop consisting of user feedback and subsequent language refinement steps. While this empirical process is still valuable, today the tasks of building and evaluating DSLs have become part of the Software Language Engineering (SLE) discipline [143] [144]. The SLE process improves dramatically the DSL creation task, as it provides a range of tools and structured methodologies [37] to help the language designer and domain expert converge their knowledge into semantics.

However, the SLE discipline still leaves important gaps for the DSL evaluation step, as mentioned by Gabriel et al [145]. Their review of a number of DSL implementations indicates a clear lack of focus on the task of systematically evaluating various verification parameters of a DSL approach such as:

- **Expressiveness**: Are the language semantics good enough to express clearly the domain?
- **Usability**: Are the tools associated to the language easy to use?
- **Effectiveness**: Can the language perform its task(s) on the domain of use for both the user and the domain expert?
- **Maintainability**: Can the language evolve and be corrected easily?

Some of these criteria can be subjective and require empirical evidence from long term use of the tools by a community of domain experts and IT professionals. The usability criterion is one of them. Others can be measured more objectively. For instance, the expressiveness and effectiveness criteria are largely correlated to the ITPSL functional requirements of section 4.3 .

An additional point to emphasize is that due to the close relation between the language semantics and the audit record (section 5.1), a complete benchmark of the language should measure the expressiveness and effectiveness of both the audit record and the language semantics. ITPSL can detect or predict only what LUARM can capture. Hence, one of the important questions to answer is whether the devised audit record format is expressive and effective enough to capture data for a range of IT misuse scenarios. Consequently, this means that the project needs data from real IT misuse incidents.

In order to address the data generation issue, the project made the source code of LUARM publicly available [82] in May 2010, hoping that early user adopters can provide data. Finding a suitable user audience willing to test the LUARM/ITPSL tools and provide real data proved to be difficult for a number of reasons. The first of these reasons relates to the level of information sensitivity of an insider misuse incident for the organization. Various information security surveys [6] [7] indicate a clear reluctance of the respondents to admit the occurrence of insider IT misuse incidents, fearing reputational damage. Apart from the lack of suitable IT misuse expression standards that this research project is trying to address, this reluctance is another important reason that contributes to the lack of data repositories for insider IT misuse mentioned in [57].

A secondary difficulty in collecting real IT misuse data relates to data compliance reasons. Even if we find organizations that are willing to use the audit engine and admit the occurrence of insider incidents, LUARM stores a large number of organization sensitive data (IP addresses, usernames). Organizations are usually reluctant to entrust these sensitive data to a University research group.

Despite these difficulties, the research project managed to find two commercial companies and one educational institution that tested LUARM in order to monitor legitimate user actions. Permission to publish the data was not obtained. Thus, the project needed a way to make use of these data, in order to make a reproducible data set that can be used to measure objectively the capabilities of the semantics and the audit record. One obvious approach would be to find a way to pseudonymize the data and thus hide all references to the companies and research institutions that allowed us to capture misuse data. Data pseudonymization [146] is a growing practice on a number of Intrusion Detection domains. However, the complexity of this approach and the reluctance of the organizations to let us publish those data in any form made the project follow a different approach.

The chosen approach involves a multiple step misuse detection game. The aim of this game is to produce a publicly available data set that will allow the replay of the real world misuse incidents in LUARM format for the purposes of evaluating LUARM/ITPSL. The publication of the data means that the results can be reproducible and act as a way for language users to verify and experiment with the language in similar scenarios. The game steps are outlined below:

- **Step 1: The making of text based scenarios**: A text-based scenario is made for all the recorded real misuse incidents. The scenario uses fictitious names for the description of the entities and roles such as company and user names. It only mentions relevant background details of each misuse case. This step ensures that we have no references to the original source of the recorded data and hence ensure third party confidentiality.

- **Step 2**: **The game briefing:** The scenarios are discussed amongst an analyst and a user team. The analyst is the person that re-enacts the security officer's role by using LUARM and ITPSL to monitor the actions of the user team with the aim to detect/predict the IT misuse acts described in the scenarios. The user team consists of a number of persons that will re-enact the scenarios by playing the role of the insider on a live IT infrastructure. The analyst briefs the users about the scope of the game and the misuse scenarios. He asks the user team not to disclose the person-role allocation nor the physical hosts where the misuse scenarios will take place. This last issue is vital for the credibility of the LUARM/ITPSL evaluation, by making misuse detection and prediction easier for the analyst.
- **Step 3: The setup of the live IT infrastructure:** The analyst sets up all the necessary details (LUARM client-server, making of the fictitious user accounts, verification of the network connections and all relevant scenario details) for the game to commence. The live IT infrastructure is a game requirement because it introduces a number of realism elements. The most important for them is the 'noise' (non relevant user activity data) the live infrastructure contributes to the game detection and prediction process. This can prove the storage efficiency of the LUARM engine, as well as the filtering capabilities of ITPSL/LUARM to mine relevant user actions from a large collection of data.
- **Step 4: The game play:** The LUARM recording starts and the user team is informed to go ahead and perform the scenarios at their convenience within a previously agreed date, in order to keep the logs at a reasonable size for publication.
- **Step 5: The analysis of data:** The LUARM monitoring stops and the analyst starts making misuse signatures to detect the misuse scenarios using the ITPSL compiler and the LUARM

logs. This is the step where we evaluate LUARM/ITPSL by answering the following questions:

- Were the language semantics good enough to express all aspects of the scenarios (expressiveness)?
- In the process of expressing the scenarios, did the language meet the ITPSL functional requirements of section 4.3?
- Did the language and the audit record allow detection and prediction of all the scenarios in question?

- **Step 6: Post curation of data:** Due to the fact that the game was recorded on a live IT infrastructure, all non fictitious user ids and other types of data (sensitive command line arguments) are pseudonymized, prior releasing the data set to the public.

Section 7.3 presents the actual scenarios and the results of the game. The next section then presents the final piece of the puzzle, the ITPSL compiler.

7.2 The ITPSL compiler

Figure 6.6 of Section 6.3 presented the ITPSL XML components. All of the these modules together with some additional logic are called collectively as 'the ITPSL compiler'. Figure 7.1 provides a different view of these components and illustrate the modules of the ITPSL compiler prototype system.

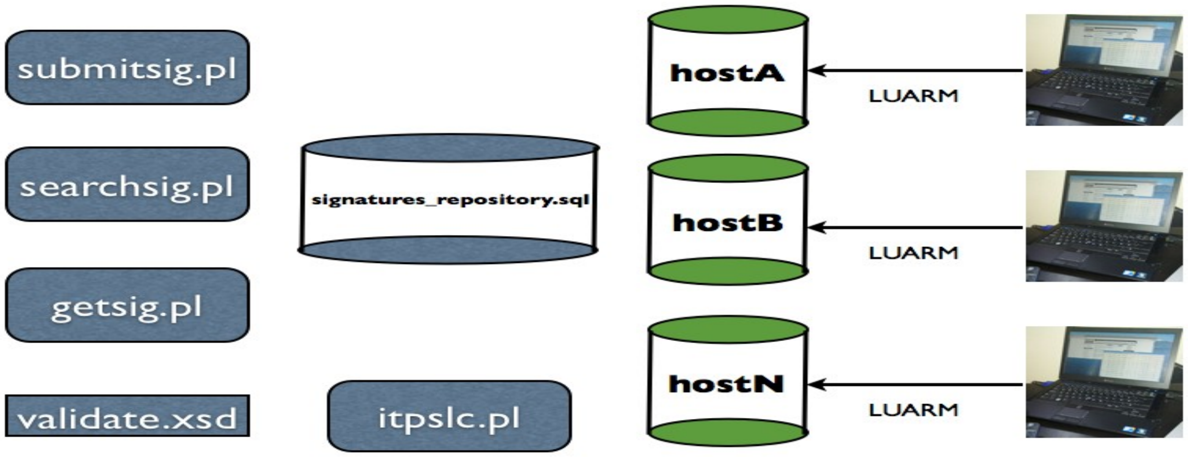


Figure 7.1: The ITPSL compiler

The prototype system consists of a number of relational tables and Perl scripts, as well as an XML Schema [133] file (validate.xsd). This file describes the syntax of the ITPSL signature and helps the various tools ensure that the processed signature is consistent with the syntax and format of the language, as described in Chapter 6. On the left hand side of Figure 7.1, the 'submitsig.pl' script helps the user submit a syntactically correct signature to the ITPSL signature repository ('signatures_repository.sql'). This repository implements the ITPSL signature ontology (ITPSL signature header). In contrast, the 'searchsig.pl' and 'getsig.pl' are used to search and retrieve one or more signatures from the ITPSL signature repository.

The fetched signatures are then fed to the main ITPSL compiler module ('itpslc.pl'). This module has the vital job of interpreting LUARM data by turning the ITPSL signature semantics into misuse detection and prediction output, informing the analyst on whether something is happening (misuse detection) or is about to happen (threat prediction). Thus, the output of the compiler may have two forms:

- **one or more detection results of the form: (user_name, 0/1):** expressing false or true outcome for detecting a misuse scenario for one or more user accounts in one or more hosts.
- **one or more predictive results of the form: (user_name, EPMO):** as described in section 6.4 (Figure 6.8), an EPMO expresses a likelihood of a misuse occurrence associated to a user name.

At functional level, the ITPSL compiler issues two types of queries:

- **LUARM-SQL queries**: This query type mines information about a range of archived events (file, network endpoint, process execution) from each of the LUARM client information repositories.
- **Live client queries**: These queries concern information that needs to be captured from at the present point of time. This is the job of the Live Client Query Forwarder functions implemented inside the main compiler module ('itpslc.pl') that talk directly to the LUARM clients.

The type of ITPSL directives employed to describe a scenario dictate which of the two query types is going to be issued. For example, most of the user/group ability access statements ('usercaneccessdir', 'usercanaccessnet') as well as statements like 'fileexists', 'direxists' and 'netexists' could generate live client queries, whereas ITPSL statements such as 'fileaccess' and 'netaccess' generate LUARM-SQL queries. A combination of these two different types of statements inside an ITPSL signature sub-block (Figure 6.11, Section 6.4) can be used to check whether something that happens now (live client query) can be correlated to an event that happened previously for which evidence is no longer present in the system (LUARM-SQL query).

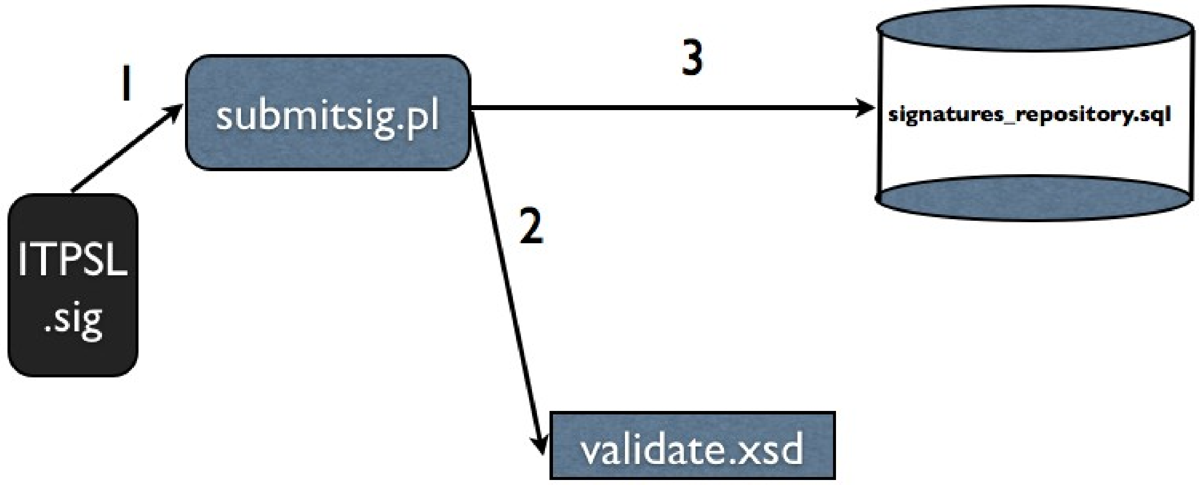


Figure 7.2: ITPSL repository signature submission

In order to better understand the function of the prototype system, it is useful to illustrate the series of actions that occur amongst the various modules during certain system operations. Figure 7.2 displays the action series of a signature submission. The 'submitsig.pl' accepts a single file as its input which contains the ITPSL signature markup (1). It then checks to see whether the submitted ITPSL markup follows certain syntactic rules, as specified in the XML Schema file 'validate.xsd' (2). If the submitted markup is syntactically valid, some additional logical validity checks are performed on the signature to ensure it is fit for submission. If anything fails at this stage, the script will halt the operation and inform the user about the syntax/logical error detected in the submitted signature. The final action (3) involves a check on whether the signature has been submitted before to the repository. If that is not the case, the signature is then entered successfully into the ITPSL repository.

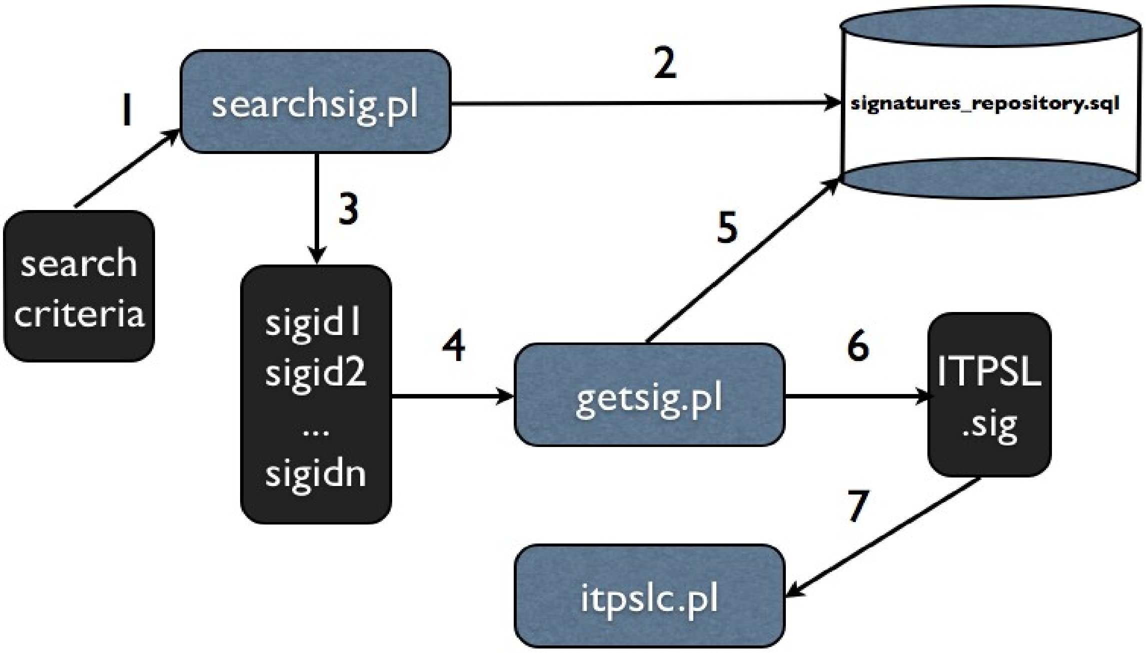


Figure 7.3: Search, retrieve and process an ITPSL signature from the repository

Once a signature is successfully entered into the repository, it may be retrieved in order to be fed to the ITPSL compiler. Figure 7.3 displays the series of actions taking place to retrieve a signature and pass it to the 'itpslc.pl' module for processing. The first step (1) is to search for one or more signatures using a range of search criteria (example: keywords, hostlist), as described by the signature ontology header (Figure 6.9, Section 6.4). These search criteria are fed to the 'searchsig.pl' module which performs a lookup on the signature repository (2). The module will return zero, one or more signature identifier (sigid's) that match the search criteria (3). A 'sigid' is the unique identifier of a signature entry in the ITPSL repository and can be passed as an argument to the 'getsig.pl' module (4). After performing the lookup of the entered 'sigid' (5), the module will return a complete signature (6). The user can then examine the signature and submit it to the main compiler (7) for processing.

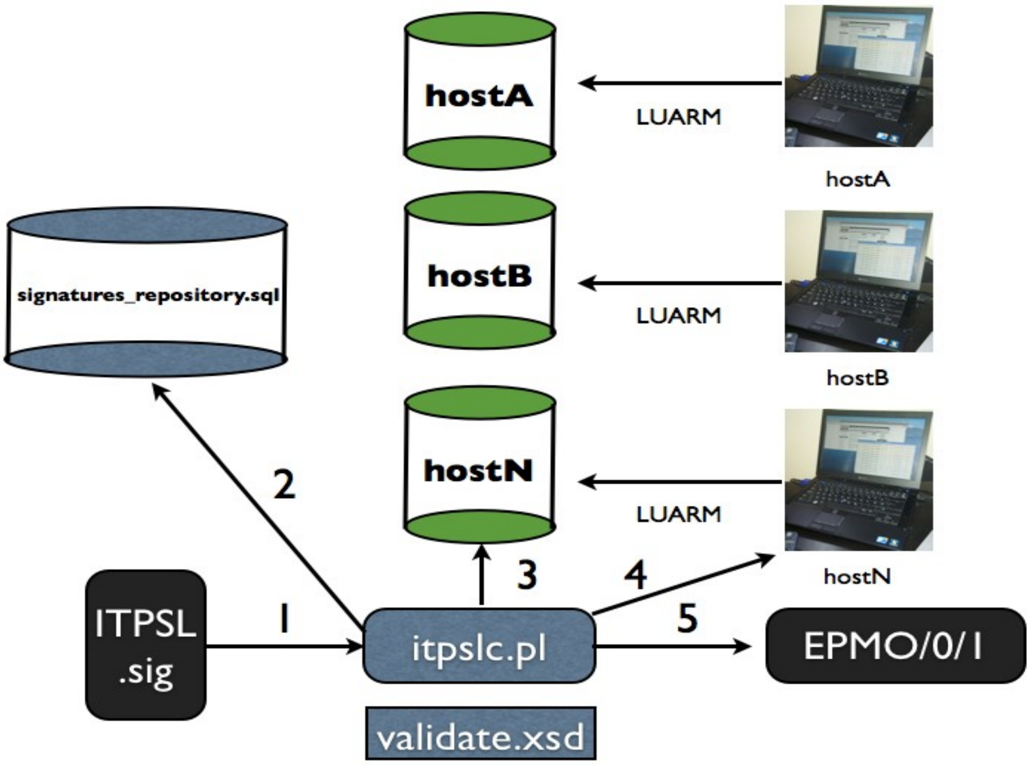


Figure 7.4: The ITPSL compiler in action

Figure 7.4 shows the module interactions when a signature is submitted to the main ITPSL compiler (1). The ITPSL signature will run again a range of syntactic checks similar to the ones performed by the 'submitsig.pl' module (Figure 7.2). It will then search the signature repository to lookup the LUARM database name for the hosts involved in the signature (2). Once the host names are located, steps 3 and 4 represent groups of LUARM-SQL and live client queries respectively, as dictated by the signature directives. The results are assembled together and the very final step is the compiler output (5).

These prototype modules represent a set of language tools with which one could interact with the ITPSL semantics. Appendix B contains more practical information about the implementation and use of the ITPSL compiler. The next sections demonstrate how these tools can be put into use against a range of insider misuse scenarios.

7.3 The Insider Misuse game

Section 7.1 described the methodology for assessing ITPSL as a DSL language and proposed a multiple step insider misuse detection game for that purpose. After presenting the language and the tools that implement it, this section discusses the insider misuse detection game in detail and explains its design rules.

The game is a controlled experiment on insider IT misuse detection and prediction and has three important entities:

- **The users**: The people that are associated to a particular scenario and have unique user-id and authentication credentials to individual workstations in the IT infrastructure. Some of these users are responsible for performing the misuse act.
- **The analyst**: The person who is responsible for examining the logs and using LUARM/ITPSL to detect/predict the threats and the associated users. In a real world scenario, this could be the security officer or a third party security auditor.
- **The IT infrastructure**: It consists of a number of hardware and software components that are setup to simulate insider threat scenarios:

- Linux workstations that run the LUARM client processes. Users have access to these workstations at various levels (simple user, admin).
- The data network that consists of Ethernet switches that interconnects all components.
- The LUARM audit engine server that contains the audit data and signature repositories the ITPSL tools to construct signatures for insider misuse detection/prediction.

Three important basic assumptions about the experiment should be stated:

- **Assumption 1**: No users have access to the LUARM audit engine server or can corrupt/disrupt the operation of the audit engine. The analyst is the only person that controls the LUARM server. In a real world scenario, this assumption is unrealistic because logging systems are software systems. Thus, they exhibit a number of security vulnerabilities themselves. However, the goal of this research is to demonstrate the usefulness and weaknesses of the LUARM audit record and the ITPSL markup, in the process of specifying insider threats. Therefore, the prototype LUARM system was designed to do exactly that and nothing more. Chapter 8 will discuss ways to strengthen LUARM for real world deployment.

- **Assumption 2**: The scenarios were derived by real world LUARM captured data from a number of organizations that adopted the project's audit engine for testing purposes. However, permission to reference the LUARM adopters and publish the original audit data was not obtained by the organizations in question. As a result, careful re-enactment of the real world incidents by a team of users under a controlled IT infrastructure is the only way to publish test results. Detecting and predicting insider IT misuse incidents in real world conditions is certainly a more laborious and challenging task than re-enacting summaries of incidents in a computer lab. Nevertheless, a careful summary and replay of incidents can highlight the pros and cons of the LUARM/ITPSL design.

- **Assumption 3**: In order to increase the realism of the game to a maximum possible extent, the users do not communicate their actions to the analyst during the game. In addition, the experiment introduces a number of hosts and possible user accounts for each scenario, so that the possibilities of performing a misuse act increase. This means that it becomes more difficult for the analyst to predict/detect where (physical workstation), when (the time of detection) and who (the name of the simulated user) performed the misuse act.

The following paragraphs provide the scenario descriptions. Each description provides a set of user names associated to the user scenario. The scenario usernames act as account decoys, as previously stated in Assumption number 3.

- **Scenario 1**:'Autobrake' Corp is a company designing car braking systems. Their engineering department is the most information sensitive work area. The braking system design process takes place, in high performance Linux workstations, one for each design engineer. The engineers have normal user rights to the workstations. Superuser rights (root) is given only to the IT admin. The designs reside on the local hard drives of the workstations and the company's IT policy forbids any transfer of sensitive data to portable media. Autobrake's system administrator has requested a salary raise various times. This has been denied by management, as the company faces a declining car manufacturing market. The system administrator is lured by a competing company that asked him to deliver schematics of the new and revolutionary Autobrake's RGX9 SUV braking system in return for a large amount of money. Enjoying the trust of everyone and having full control of the engineering CAD workstations, the system administrator decides to take the offer of the competing company. He performs the intellectual property theft by following a well designed approach which is summarized below:

- He carefully chooses the user account of a mechanical engineer (username 'engineer3') that had some disputes over work issues with management. He aims to avoid detection by means of masquerading as the engineer in question.

- After successfully masquerading as the engineer in the IT system he uses a portable USB key to obtain the commercially sensitive RGX9 schematic, leaving only the traces of the user engineer “actions”.

(scenario user accounts: root, engineer1, engineer2, engineer3).

- **Scenario 2**: 'Agrico' is a small company (20 people) located in the countryside. It works as a price broker for various farming fertilizers. The company employees need access to the World Wide Web in order to search for on-line prices on various vendor web sites. They use the online web data to make an up-to-date 'best price' database that they sell to their customers. Agrico's IT manager has a low budget to cater for the internet connection. Their only Internet access option (due to the rural location) is a 2 Mbps ADSL line. The ADSL connection has recently become slow and various brokers complain to the IT manager about slow access speeds. After some investigation, a fault with the ADSL line is ruled out and a cheap proxy cache server is installed, in an attempt to reduce the bandwidth requirements. The proxy logs indicate to the IT manager that various pornographic websites are being visited and they find in the proxy server file space various high definition pornographic videos and photos, making clear that they are the reason the ADSL connection is saturated. The proxy cache logs are in the following form:

Date/Time reference	**Workstation IP**	**URL**
18/12/2010-14:13:23	**192.168.0.107**	**http://mybadsite.com**

As a result, the CEO issues an email to all employees explicitly forbidding access to such sites. In an attempt to mitigate the situation, the IT manager tries to block the URLs of these sites at the proxy server. However, he discovers that over time, new web sites are being visited. As all employees have been warned that their online actions are being monitored, the CEO orders the IT manager to locate the offenders and provide evidence of their actions. The first action of the IT manager is to try and correlate the data from the proxy logs to the login activity of the workstation. This gave him a username, however, web browser history and cache provided no hits. As each of the workstations might be used by different users, this breaks the chain of evidence. We see how ITPSL and LUARM can help. (Scenario user accounts: agrico1,agrico2,agrico3).

- **Scenario 3**: John (username:johnc) is a talented software developer in the central IT services of a known University. However, he has strong problems with authority and his liberal personality has caused some problems with the IT services manager and various people at his workplace. John has frequently argued that he should be allowed to use a Bit Torrent protocol in his development workstation, even if the the IT regulations explicitly forbid the installation of Bit Torrent software in University computing devices. Apart from the use of Bit Torrent software that is tolerated by the network administrators, John has recently got in trouble by installing the LOIC (Low Orbit Ion Cannon) DoS client. While the intentions of John were unknown, the network manager believes that he intended to DoS various Amazon and Paypal servers, in response to the Wikileaks controversy. This shows how LUARM and ITPSL can help to mitigate these situations.

- **Scenario 4**: Detection of unintentional misuse: The Human Resources department of Autocorp has a file share called 'Payroll'. The share contains sensitive information and has the following folder structure:

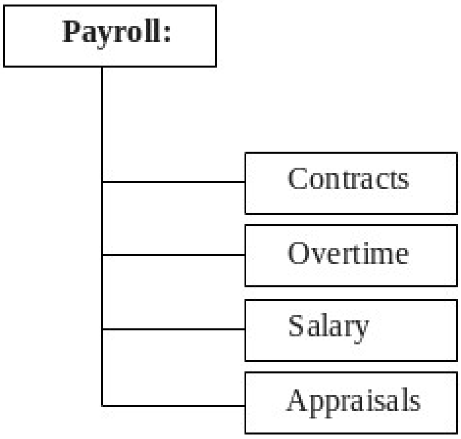


Figure 7.5: Directory structure for Scenario 4

The folder is owned by members of the user group 'hrpersons'. The 'Appraisals' subfolder is open to the entire company, so that staff can see appraisals. The 'Overtime' subfolder is accessible by all members of the 'accounts' group and 'hrpersons'. The 'Salary' and 'Contracts' are only accessible by select few people of the 'hrpersons' user group (usernames: ridh, mikes). Here, we show how ITPSL can help detect permission leakage problems that often happen by accidentally configuring permissions for contracts and salary folders being accessible by users other than ridh and mikes. Other users (not from the 'hrpersons' or 'accounts' user groups) having access to Overtime folder.

These scenarios represent a range of IT misuse incidents. Scenario 1 is a typical Intellectual Property theft scenario with the added complexity of a masquerade attack (an attack which involves an insider pretending to be another user, to perform the misuse act AND incriminate somebody else). It is designed to demonstrate how LUARM can provide a strong chain of evidence for detecting the occurrence of a past incident. The same can be said for Scenario 2 which re-enacts the detection of accessing pornographic material, a common incident according to information security surveys.

Scenario 3 is an example of an insider IT misuse prediction task. In essence, it aims to demonstrate the decision theoretic information features of ITPSL and help the analyst predict the installation of a dangerous DoS attack tool by an ordinary user. The fourth and last scenario of the game is also demonstrating a predictive operation of file access control settings that could produce accidental (non intentional) information leak.

The scenarios were discussed during an initial briefing amongst the analyst and three IT specialists that would re-enact these incidents by playing the role of the misuser for each scenario. After the initial briefing, the users met amongst them to discuss a role allocation for each scenario without the knowledge of the analyst. At that point, LUARM was activated and logging commenced for a period of 4 weeks. After the four week audit period, results were collected and verified with the users by the analyst.

After explaining the basic assumptions and role of the game, the next section demonstrates the use of LUARM/ITPSL in the first game scenario.

7.4 Detecting intellectual property theft

This section discusses how LUARM was used to provide accountability and locate the misuser for the intellectual property theft scenario (Scenario 1, Section 7.3). Scenario 1 is a complex scenario that requires strong chain of evidence to incriminate the offender. A post-mortem forensic examination style of an incident is provided, in order to demonstrate the correlation power of the LUARM audit record structure.

The investigation begins at the MySQL LUARM server side, where the detection of relevant file activity took place. In particular, instead of shifting manually through text log files and use information filter tools such as grep, awk and sed [147] to locate activity relevant to the prototype file 'RGX9', a series of LUARM-SQL statements is issued. From the audit records of various workstations, LUARM indicates that the workstation with name 'proteas' contains many hits for the file RGX9:

```
mysql> select COUNT(*) from panoitpsl.fileinfo where filename RLIKE 'RGX9' OR location RLIKE 'RGX9'  \G

*************************** 1. row ***************************

COUNT(*): 0

1 row in set (0.05 sec)
```

```
mysql> select COUNT(*) from dionitpsl.fileinfo where filename RLIKE 'RGX9' OR location RLIKE 'RGX9'  \G

*************************** 1. row ***************************
COUNT(*): 0
1 row in set (0.05 sec)
…
mysql> select COUNT(*) from protitpsl.fileinfo where filename RLIKE 'RGX9' OR location RLIKE 'RGX9'  \G

*************************** 1. row ***************************
COUNT(*): 145
1 row in set (0.05 sec)
```

Thus, by loading the 'proteas' workstation LUARM database, the detected 145 hits can be further examined :

```
mysql> use protitpsl;

Database changed

mysql> select username,pid,cday,chour,cmin,location,filename from fileinfo where filename RLIKE 'RGX9' OR location RLIKE 'RGX9' \G
```

From the many resulting hits, the following ones stand out:

```
*************************** 111. row ***************************
username: engineer3
pid : 8301
cday: 4
```

```
chour: 15
cmin: 30
location: /storage/users/engineer3/work/designs
filename:RGX9.jpg
...
*************************** 118. row ***************************
username: engineer3
pid: 28538
cday: 4
chour: 15
cmin: 32
location: /media/U3SAN03-12
filename: RGX9.jpg
```

The reason these file access patterns looked suspicious is that they were different to the normal pattern of accessing the file by the staff engineer. Normally, user 'engineer3' would access the file by means of certain design and image editing applications, under its usual directory (/storage/users/engineer3/work/designs). This time, however, things look a bit different, if one follows the association of file access to process execution, in order to confirm which programs performed the recorded file transactions.

```
mysql>select username,pid,command,arguments,cyear,cday,chour,cmin from psinfo where username='engineer3' AND pid='8031' AND cyear='2011' AND cday='4' AND chour='15' AND cmin='30';
```

```
*************************** 1. row ***************************
username: engineer3
pid: 8031
command: /bin/cp
```

```
arguments: work/designs/RGX9.jpg /tmp/
cyear: 2011
cday: 4
chour: 15
cmin: 30

mysql>select username,pid,command,arguments,cyear,cday,chour,cmin from psinfo where username='engineer3' AND pid='8031' AND cyear='2011' AND cday='4' AND chour='15' AND cmin='30;

*************************** 1. row ***************************
username: root
pid: 28538
command: mv
arguments: RGX9.jpg /media/U3SAN03-12
cyear: 2011
cday: 4
chour: 15
cmin: 32
```

Essentially, the previous results verify that the file was first copied from the normal directory to /tmp and then was moved to the /mnt/usb. At this point, a little bit of system specific knowledge comes into light, as /mnt/usb is the usual mount point where Linux links portable storage media to the filesystem. Hence, the question to raise is whether a portal storage medium was connected to the workstation, prior to the 'mv' file transaction. The query result yields a positive answer:

```
mysql> select * from hwinfo where cyear='2011' AND cmonth='01' AND cday='04' AND chour='15'\G
*************************** 1. row ***************************
hwdevid: 71
```

```
md5sum: a16e7386f14de769a7a9491da2071f5b
cyear: 2010
cmonth: 12
cday: 4
chour: 15
cmin: 30
csec: 28
devbus: USB
devstring: Cruzer Micro U3
devvendor: SanDisk Corp.
userslogged: engineer3,root
dyear: 2010
dmonth: 1
dday: 4
dhour: 15
dmin: 33
dsec: 38
```

This database hit seems to be in line with the actions of engineer3, as it indicates a device connection before the execution of the 'mv' command and a disconnection well after the mv command.

Thus, everything seems to point out that 'engineer3' violated the company policy and transferred a sensitive file to a USB medium, against the company IT regulations. However, this had been categorically denied by the actual person in the real world case. Moreover, a good but non IT based alibi for the staff engineer (again from the real world case) was that he exited the building with his security card token around 14:50, returning back to his desk at 15:50, a wide gap for him. Clearly, something else was going on and the clue was the 'userslogged' field of the last LUARM result.

This 'hwinfo' LUARM table field contains the usernames for accounts that are logged into the workstation at the time of the device connection. Apart from 'engineer3' we note the root account being active, which is clearly the only other choice that, under the circumstances, could have performed the mount procedure.

Based on the time stamp of the mv operation, a careful investigation of the root account actions reveals a key command execution, derived from the 'psinfo' table:

```
mysql> select * from psinfo where pid='27865' AND cyear='2011' AND cday='4' AND cmonth='1' AND
chour='15' AND cmin >= '20' AND cmin <='33' \G
*************************** 1. row ***************************
psentity: 97654
md5sum: 7067284f2e1aefc430339ef091b4e41b
username: root
pid: 27865
ppid: 26407
pcpu: 0.0
pmem: 0.0
command: su
arguments: - engineer3
cyear: 2011
cmonth: 1
cday: 4
cmin: 28
chour: 15
csec: 36
dyear: 2011
dmonth: 1
```

dday: 4

dhour: 15

dmin: 28

dsec: 39

The 'su' command is used routinely by administrators to switch user credentials, in order to test environment settings and perform system tasks [14]. However, it can be easily used as a masquerading tool to covertly perform actions using the credentials of somebody else.

A further investigation also found the USB key on the desk of the IT administrator with the RGX9.jpg file. The hwinfo table device identifier data ('devstring', 'devvendor') as well as the mount point identifier (/media/U3SAN03-12) from the psinfo commands contributed towards strengthening the final piece of the puzzle.

This case shows the versatility of the relational structure of the LUARM record that paved the way from simple file operation to related program execution and other events that can provide strong evidence and lead to the misuser.

7.5 Detecting pornography browsing during working hours

Scenario 2 simulates a real world IT misuse incident which involved the detection of a person that -contrary to the company IT usage policy- used the Internet connection to access pornographic content during working hours. The abundance of pornography on the Internet makes this type of IT misuse a quite common problem with profound economic and productivity implications for the affected organizations [148]. Hence, it is useful to see how LUARM and ITPSL can tackle the detection of this problem and provide evidence of the offender.

The investigation starts by looking into the proxy server logs. As previously mentioned by the scenario description, the proxy server logs indicate more than a hundred hits on URL's that were clearly referring to web sites were visited by the workstation with IP 192.168.0.107 (slart) almost on a daily basis. The proxy log for the 18th of December 2010 (one of the days the particular IT misuse incident took place is included below:

```
Date/Time reference          Workstation IP          URL
18/12/2010-14:13:23          192.168.0.107           http://mybadsite1.com
18/12/2010-14:14:37          192.168.0.107           http://mybadsite1.com
...
18/12/2010-14:21:32          192.168.0.107           http://mybadsite2.com
...
18/12/2010-17:37:28          192.168.0.107           http://mybadsite2.com
18/12/2010-17:44:48          192.168.0.107           http://mybadsite1.com
```

However, as this workstation is used by many users throughout the day simultaneously, it is not clear who accessed the pornographic sites. The proxy logs are an indication of the problem's presence, however they do not constitute concrete evidence of a user accessing the sites. One needs to query the LUARM records for host 'slart', in order to obtain definite evidence of user activities. This is when the first problem with the LUARM audit record structure is encountered:

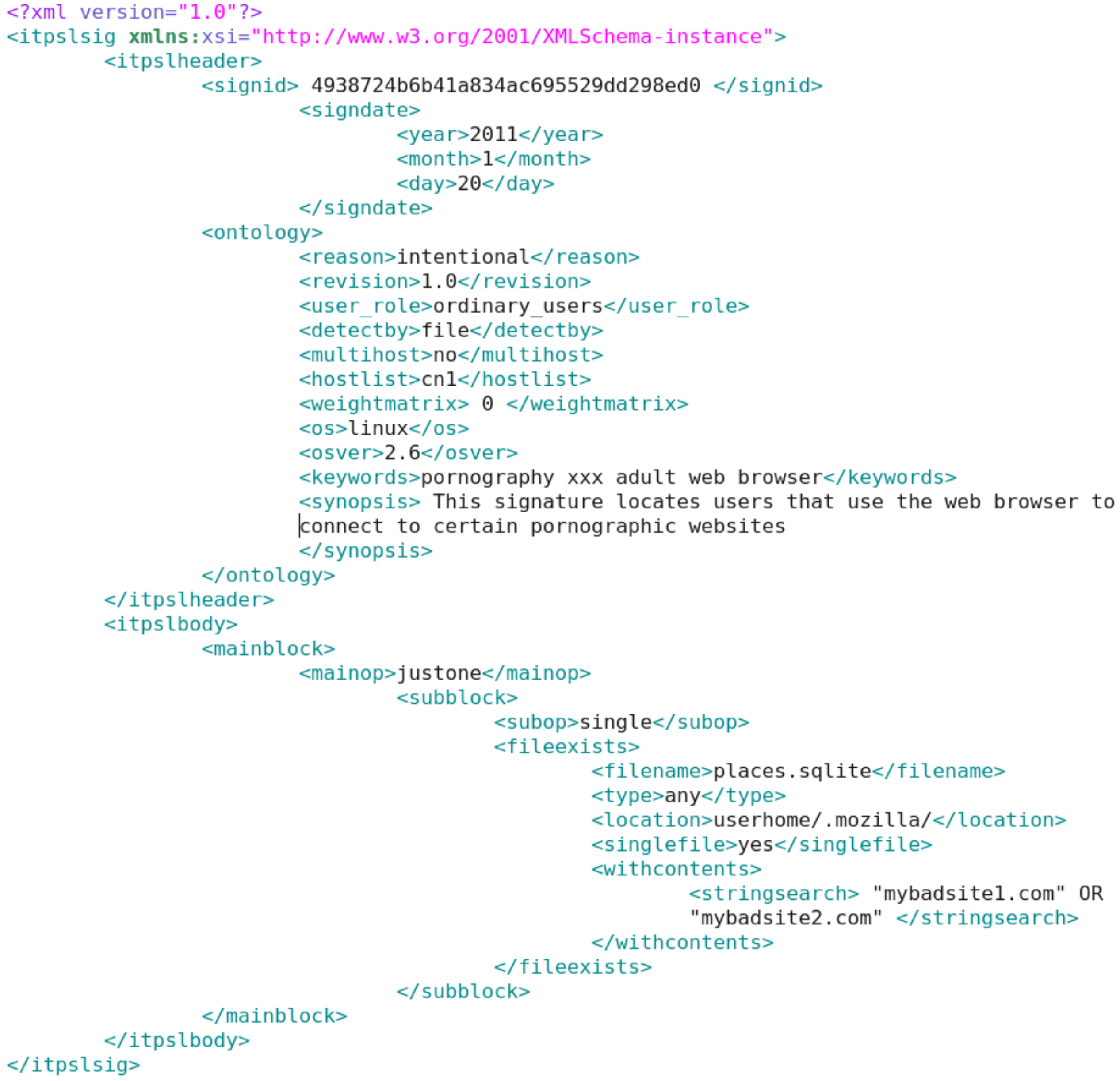
```
mysql> use slartitpsl;
Database changed
```

```
mysql> select COUNT(*) from netinfo where application RLIKE 'firefox' AND cyear='2010' AND cday='18'
AND cmonth='12' AND  destport='80' AND destfqdn RLIKE 'mybadsite1.com'  LIMIT 100 \G
*************************** 1. row ***************************
COUNT(*): 0
1 row in set (0.01 sec)
mysql> select COUNT(*) from netinfo where application RLIKE 'firefox' AND cyear='2010' AND cday='18'
AND cmonth='12' destport='80' AND destfqdn RLIKE 'mybadsite2.com'  LIMIT 100 \G
*************************** 1. row ***************************
COUNT(*): 0
1 row in set (0.00 sec)
```

In this case, LUARM was queried to return any detected hits on the host 'slart' as a result of the firefox web browser application accessing port 80 (http) of 'mybadsite1.com' and 'mybadsite2.com'. It returned zero results. The query was repeated for other browsers (konqueror, chrome) but still no results were returned, something which appeared to be alarming at first sight.

A more careful examination revealed the culprit. LUARM records only the IP addresses of the endpoints. Chapter 5 discussed the reasons behind this design choice. As many adult websites do not provide reverse DNS name resolution [98] creates a situation where the audit record cannot resolve the recorded IP to a Fully Qualified Domain Name (FQDN). Malicious spammers and phishers use a similar technique to avoid detection [149]. Whether adult website owners intentionally or accidentally do not configure reverse DNS is unclear. However, the end result is that this incapacitates LUARM to tag the recorded IP with the URL that hints a pornographic sight.

This is a serious problem that hinders the detection process. It can be circumvented by knowing alternative places to look for, in order to find signs of the web proxy recorded URLs. In this case, it is known that the Firefox web browser keeps a record of visited pages inside the file places.sqlite [150], located under the home profile of the user. Other web browsers have similar settings to record the web pages visited by the user.



```
<?xml version="1.0"?>
<itpslsig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <itpslheader>
                <signid> 4938724b6b41a834ac695529dd298ed0 </signid>
                        <signdate>
                                <year>2011</year>
                                <month>1</month>
                                <day>20</day>
                        </signdate>
                <ontology>
                        <reason>intentional</reason>
                        <revision>1.0</revision>
                        <user_role>ordinary_users</user_role>
                        <detectby>file</detectby>
                        <multihost>no</multihost>
                        <hostlist>cn1</hostlist>
                        <weightmatrix> 0 </weightmatrix>
                        <os>linux</os>
                        <osver>2.6</osver>
                        <keywords>pornography xxx adult web browser</keywords>
                        <synopsis> This signature locates users that use the web browser to
                        connect to certain pornographic websites
                        </synopsis>
                </ontology>
        </itpslheader>
        <itpslbody>
                <mainblock>
                        <mainop>justone</mainop>
                                <subblock>
                                        <subop>single</subop>
                                        <fileexists>
                                                <filename>places.sqlite</filename>
                                                <type>any</type>
                                                <location>userhome/.mozilla/</location>
                                                <singlefile>yes</singlefile>
                                                <withcontents>
                                                        <stringsearch> "mybadsite1.com" OR
                                                        "mybadsite2.com" </stringsearch>
                                                </withcontents>
                                        </fileexists>
                                </subblock>
                </mainblock>
        </itpslbody>
</itpslsig>
```

Figure 7.6: ITPSL signature to detect IT misuse for Scenario 2

Locating the files that contain these criteria is a task that is best left to the ITPSL compiler. Rather than trying to manually search all the user browser history files for the URL of the offensive web-sites, one can make a signature and use it to shift through the information collected LUARM using ITPSL semantics. Figure 7.6 shows the complete ITPSL markup to perform this action. The ITPSL directives are highlighted.

In the signature header, a single value of zero on the weightmatrix directive will force the compiler to switch in misuse detection mode. In the hostlist header directive, the value of 'cn1' represents the server which contains the user home directories (in corporate environments, it is common that user home directories are centrally located on file servers and then made available to user workstations by means of distributed file systems such as NFS [151], CIFS [152] or other protocols). The ITPSL body consists of a single statement which searches for the URL strings using an inclusive OR binary operator (in case only one of them is recorded).

The signature is submitted to the ITPSL repository for subsequent reuse, as described in Figure 7.2. Pending a successful validation, it is then fed to the ITPSL compiler in order to detect the offender. These steps take place on the LUARM server side, as shown below:

```
[root@cn1 itpslcomp]# ./submitsig.pl scenario2.xml
SUCCESS: Inserted signature with MD5 hash 378389f2469486438d0c04f6b28c8c0a into the ITPSL repository.
[root@cn1 itpslcomp]# ./itpslc.pl scenario2.xml
...
(agrico1, 1)
```

The compiler takes some seconds to issue live client queries to the home directories of users on the cn1 server and the result is the detection list **(agrico1, 1)**. This means that the ITPSL compiler claims that the user 'agrico1' is the offender. The verification by manually looking into the file confirms what the ITPSL compiler detected:

```
[root@cn1 luarm]# strings /home/agrico1/.mozilla/firefox/c9hkxp7d.default/places.sqlite | grep -i mybadsite
http://www.mybadsite1.com/videos/straight/redhead-recent.htmlSEX Free Sex Movie Galleries, Good Porn
http://www.mybadsite1.com/videos/straight/amateur-rate.htmlSEX  Free Sex Movie Galleries, Good Porn
http://www.mybadsite2.com/Asian/index.html

http://www.mybadsite2.com/blog/
```

At this point, the major question to ask is whether evidence in the browser history file is enough to incriminate a user for pornographic access IT misuse. Due to the abundance of pornography [148] on the Internet, there are good chances that someone may accidentally access pornographic material [153]. Many spam messages or misspelled domain names of widely known web sites can accidentally direct a web browser to a pornographic website. Hence, what is important to examine in cases like this is whether the access pattern from the user is persistent.

The ITPSL signature compilation gave an additional crucial link: A username associated with access to these offensive websites. One can now investigate further actions related to the username 'agrico1' by going back to the LUARM SQL interface. As stated in previous paragraphs of this section, LUARM did not manage to resolve the collected destination endpoint IPs due to lack of reverse DNS data. However, now that a username is known, a DNS forward lookup (host->IP) can provide an IP address:

```
[root@cn1 itpslcomp]# host www.mybadsite1.com
www.mybadsite.com is an alias for mybadsite.com.
mybadsite.com has address 216.18.190.104
```

```
[root@cn1 itpslcomp]# host www.mybadsite2.com
www.mybadsite2.com is an alias for mybadsite2.com.
mybadsite2.com has address 76.9.26.162
mybadsite2.com mail is handled by 10 twin3.isprime.com.
```

As a result, LUARM can be used to intercept the access pattern of these websites by the username 'agrico1'. The results indicate many hits:

```
mysql> use slartitpsl;
Database changed
mysql> select COUNT(*) from netinfo where username='agrico1' AND application RLIKE 'firefox' AND
cyear='2010' AND cday='18'  AND cmonth='12' AND destport='80' AND destip='216.18.190.104'  LIMIT 100 \G

*************************** 1. row ***************************
COUNT(*): 86
1 row in set (0.00 sec)

mysql> select COUNT(*) from netinfo where username='agrico1' AND application RLIKE 'firefox' AND
cyear='2010' AND cday='18' AND cmonth='12' AND destport='80' AND destip='76.9.26.162'  LIMIT 100 \G

*************************** 1. row ***************************
```

```
COUNT(*): 9
1 row in set (0.00 sec)
```

These hits can be further expanded to reveal a repeated and persistent access pattern (some of the results are omitted to save page space):

```
mysql> select chour,cmin,pid from netinfo where username='agrico1' AND application RLIKE 'firefox' AND cyear='2010' AND cday='18' AND cmonth='12' AND destport='80' AND destip='216.18.190.104'  LIMIT 100;

+-------+------+-------+
| chour | cmin | pid   |
+-------+------+-------+
|    14 |   12 | 27605 |
|    14 |   13 | 27605 |
|    14 |   17 | 27605 |
|    14 |   18 | 27605 |
|    17 |   20 | 12665 |
|    17 |   22 | 12665 |
|    17 |   36 | 25415 |
|    17 |   37 | 25415 |
...
|    17 |   41 | 30954 |
+-------+------+-------+
86 rows in set (0.00 sec)
```

The persistent access pattern of the offensive websites indicates that this is unlikely to be an accidental access issue. This can be substantiated further by looking back at the web browser history of the file:

```
http://www.google.com/search?q=sexy+asian+women&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a#sclient=psy&hl=en&safe=off&client=firefox-a&hs=iUa&rls=org.mozilla:en-US%3Aofficial&source=hp&q=sexy+asian+women+porn&aq=f&aqi=&aql=&oq=&pbx=1&bav=on.2,or.r_gc.r_pw.&fp=ff3e2739446bc197

http://www.google.com/search?q=sex&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a

http://www.google.com/search?q=sexy+asian+women&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a

http://www.google.com/search?q=sexoasis&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a

http://www.google.com/search?ie=UTF-8&oe=UTF-8&sourceid=navclient&gfns=1&q=tube#sclient=psy&hl=en&safe=off&q=tube+sex&aq=f&aqi=&aql=&oq=&pbx=1&bav=on.2,or.r_gc.r_pw.&fp=ff3e2739446bc197
```

This indicates that many searches were performed using a search engine, in an effort to obtain the material. The same pattern repeated for other days and at that point, one can be sure that the username 'agrico1' is responsible for intentional pornographic material access.

If the user had deleted the contents of the history file, a common move by privacy conscious users and an option with many web browsers, the previously described information path could be necessitated by performing the forward DNS lookup operation straight from the web proxy data. The querying of the LUARM SQL data against the obtained IP addresses would reach the same conclusion.

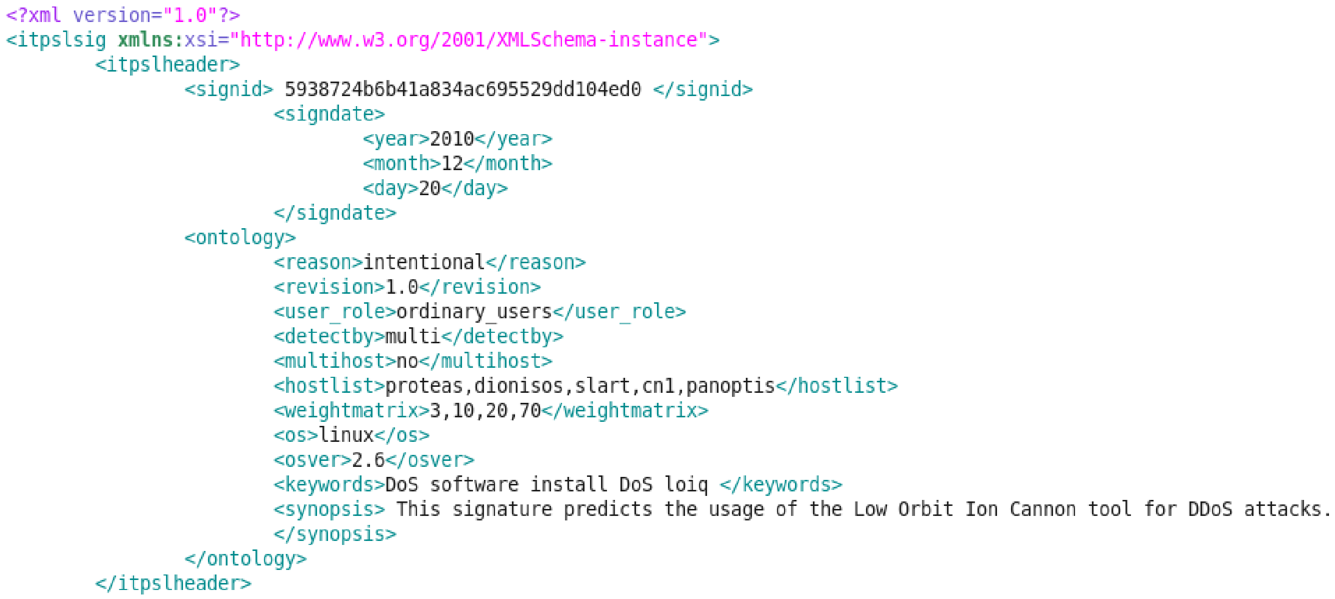
However, the fact that LUARM could not resolve the collected endpoint IPs creates a difficulty that forces the investigator to rely on external reference sources (web proxy logs). If these external references are unavailable, the only way to locate misuse is to manually go through all the non resolved endpoint IPs from LUARM records and verify that they point to offensive websites. This might be incredibly laborious and is not acceptable as a solution to the problem. Thus, this case has revealed an important issue with the design of the LUARM audit record: Although it may be good enough to hold the evidence (unresolved endpoint IPs of the audit record), this evidence becomes difficult to find. This issue needs to be addressed more thoroughly and Chapter 8 will discuss this issue.

In summary, this case has demonstrated how LUARM and ITPSL can be used in harmony to detect a common type of Insider IT misuse. The ITPSL was used to shift through the volume of LUARM data, locate crucial evidence in relation to a user name and then LUARM was used to re-enforce the evidence of IT misuse. The LUARM data from this scenario have been placed on-line for reference [82].

7.6 Predicting the installation of unauthorized software

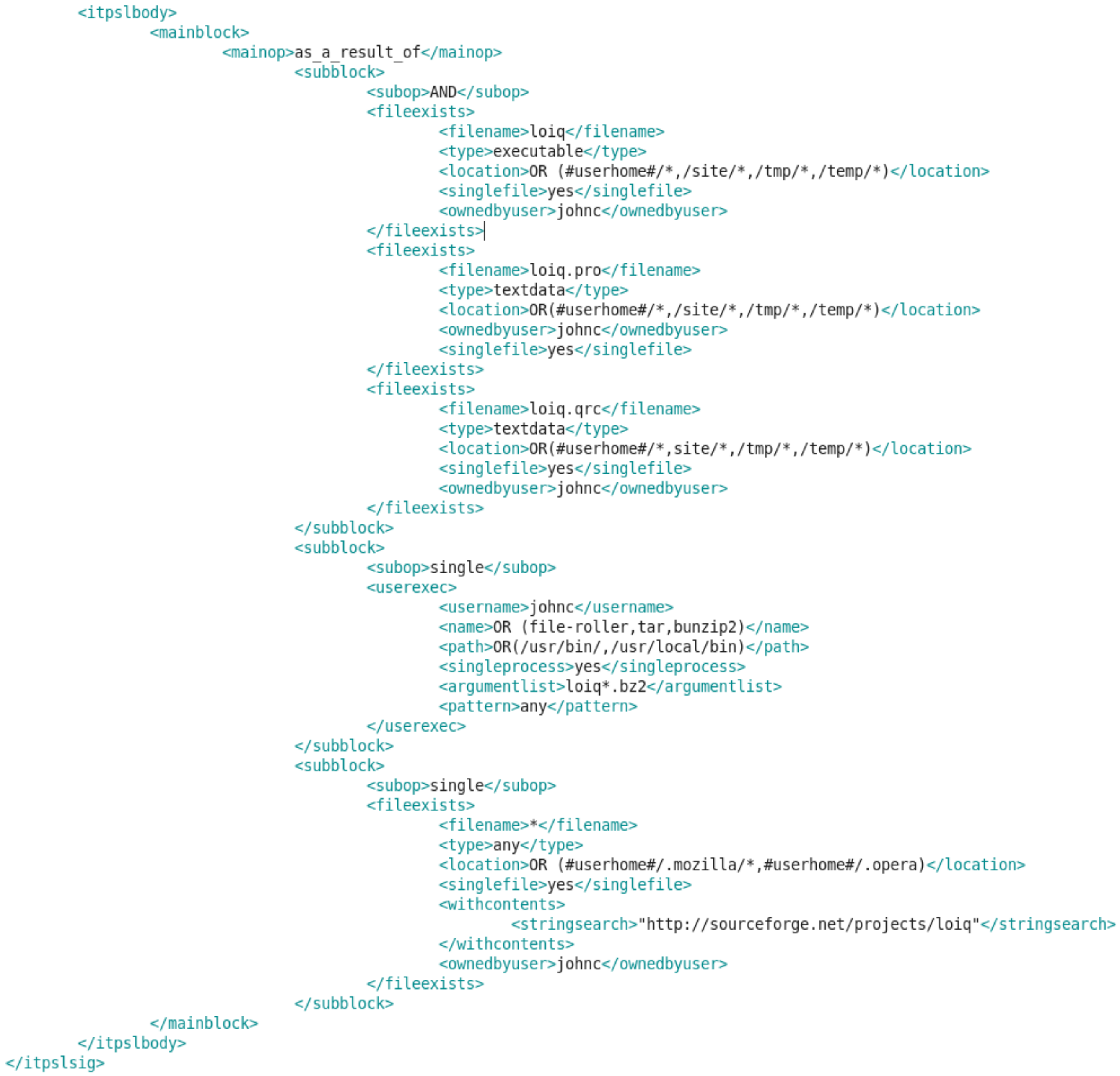
Whilst the previous scenarios were all about threat detection, the third scenario demonstrates the threat prediction capabilities of LUARM and ITPSL. Unauthorized software installation is an important threat for IT infrastructures [154] and commonly deployed operating systems have specific policy enforcement mechanisms to prevent it [155]. Beyond any prevention measures at operating system level, it is important to provide evidence of the misuse act taking place by associating the activities of the user(s) to relevant events and examine a mechanism that predicts the occurrence of the misuse act in an automated way.

This particular scenario demonstrates an insider that willingly participates into a Distributed Denial of Service Attack (DDoS) [156]. On December 8 2010, the "Anonymous" group launched DDoS attacks on organisations such as Mastercard.com, PayPal, Visa.com and PostFinance as part of the "Operation Payback" campaign [157]. This particular campaign was seeking volunteers to willingly install the Low Orbit Ion Cannon (LOIC) DDoS tool [158] and attack various websites that broke ties to the WikiLeaks organization. The scenario user 'johnc' is a potential volunteer and thus the task is to predict the installation of the software and provide enough evidence of his actions.



```
<?xml version="1.0"?>
<itpslsig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <itpslheader>
                <signid> 5938724b6b41a834ac695529dd104ed0 </signid>
                        <signdate>
                                <year>2010</year>
                                <month>12</month>
                                <day>20</day>
                        </signdate>
                <ontology>
                        <reason>intentional</reason>
                        <revision>1.0</revision>
                        <user_role>ordinary_users</user_role>
                        <detectby>multi</detectby>
                        <multihost>no</multihost>
                        <hostlist>proteas,dionisos,slart,cn1,panoptis</hostlist>
                        <weightmatrix>3,10,20,70</weightmatrix>
                        <os>linux</os>
                        <osver>2.6</osver>
                        <keywords>DoS software install DoS loiq </keywords>
                        <synopsis> This signature predicts the usage of the Low Orbit Ion Cannon tool for DDoS attacks.
                        </synopsis>
                </ontology>
        </itpslheader>
```

Figure 7.7: The header of the ITPSL signature for Scenario 3



```
    <itpslbody>
        <mainblock>
            <mainop>as_a_result_of</mainop>
                <subblock>
                    <subop>AND</subop>
                    <fileexists>
                        <filename>loiq</filename>
                        <type>executable</type>
                        <location>OR (#userhome#/*,/site/*,/tmp/*,/temp/*)</location>
                        <singlefile>yes</singlefile>
                        <ownedbyuser>johnc</ownedbyuser>
                    </fileexists>
                    <fileexists>
                        <filename>loiq.pro</filename>
                        <type>textdata</type>
                        <location>OR(#userhome#/*,/site/*,/tmp/*,/temp/*)</location>
                        <ownedbyuser>johnc</ownedbyuser>
                        <singlefile>yes</singlefile>
                    </fileexists>
                    <fileexists>
                        <filename>loiq.qrc</filename>
                        <type>textdata</type>
                        <location>OR(#userhome#/*,site/*,/tmp/*,/temp/*)</location>
                        <singlefile>yes</singlefile>
                        <ownedbyuser>johnc</ownedbyuser>
                    </fileexists>
                </subblock>
                <subblock>
                    <subop>single</subop>
                    <userexec>
                        <username>johnc</username>
                        <name>OR (file-roller,tar,bunzip2)</name>
                        <path>OR(/usr/bin/,/usr/local/bin)</path>
                        <singleprocess>yes</singleprocess>
                        <argumentlist>loiq*.bz2</argumentlist>
                        <pattern>any</pattern>
                    </userexec>
                </subblock>
                <subblock>
                    <subop>single</subop>
                    <fileexists>
                        <filename>*</filename>
                        <type>any</type>
                        <location>OR (#userhome#/.mozilla/*,#userhome#/.opera)</location>
                        <singlefile>yes</singlefile>
                        <withcontents>
                            <stringsearch>"http://sourceforge.net/projects/loiq"</stringsearch>
                        </withcontents>
                        <ownedbyuser>johnc</ownedbyuser>
                    </fileexists>
                </subblock>
        </mainblock>
    </itpslbody>
</itpslsig>
```

Figure 7.8: The body of the ITPSL Scenario 3 signature

Figure 7.7 displays the header of the ITPSL signature for the demonstration scenario. We will use multiple criteria to predict the running of the LOIC DDoS tool (<detectby>multi</detectby>). A combination of <multihost>no</multihost> and a system hostlist forces the compiler to examine the

ITPSL body specified conditions in each of the host separately. The Weight Matrix consists of the 3 elements with relative weights of 10, 20 and 70 respectively. Later paragraphs discuss the choice of those weights, however, the fact that the weightmatrix tag consists of a non-zero value suggests that the ITPSL compiler will interpret the signature in threat prediction context.

Figure 7.8 displays the body of the same signature. There are three sub-blocks inside the main-block that define three separate prediction event criteria. The main-block 'as_a_result_of' operator suggests that these events should form a sequence of three successive steps. The very top one is the final step (Figure 6.11). The defined predictive event sequence reflects three steps that a user should take **prior** executing the LOIC DDoS tool:

1. Search for the tool in the Internet and visit the tools web page: As this tool is not part of a standard Linux Operating System utility toolkit, the most probable source for getting access to the tool is to download it from the Internet. Hence, an interaction of the web browser with LOIC tool website should be one of the early steps that one should look for.
2. The decompression of the program utility archive file, as it is offered from the LOIC web site: This is the second logical step that a user should execute before he runs the utility. Most websites offer a program in the form of a compressed archive. This is the case with the LOIC tool [158].
3. If the decompression is successful, the very final step should verify the presence of certain files that are part of the contents of the compressed archive. The uncompressed files should mean that the user is ready to execute them and this is the limit between threat prediction (inferring the running of the utility as the next step) and threat detection (seeing the execution of the utility against the target websites).

Starting with the final step/event (the first subblock in Figure 7.8), a 'fileexists' ITPSL directive specifies three files that should be part of the DDoS installation archive: loiq, loic.pro and loic.qrc. These three files exist in many versions of the LOIC DDoS archive (loicVERSION.tar.bz2) contents, so their presence constitutes a distinguishing characteristic. The AND sub-block operator (<subop>AND</subop) makes sure that all three of them must exist. The OR operator adds polymorphic properties to the location of the specified files. It is not known in advance where the user will decompress the executable, so the search for the files will be performed in any path under the user home directory (#userhome#), the /site/ , /tmp and /temp locations, where it is known that the user may have write privileges.

The second sub-block defines the interim step of decompressing the LOIC DDoS tool archive. A 'userexec' ITPSL directive examines whether user 'johnc' has executed a set of possible archive decompression applications (file-roller, tar,bzip2) from different potential source paths (/usr/bin, /usr/local/bin). The filename uses a wild card (loiq*.bz2) to make the signature applicable to different versions of the LOIC utility (loiq-0.3.tar.bz2, loiq-0.2.3-1.tar.bz2, etc). Note that the temporal specification is defined as 'any' (<pattern>any</pattern>). This means that we do not bind this step to a particular time. However, as the event is part of a 'as_a_result_of' main-block operator set of ITPSL sub-blocks, it will have to occur before the final event and after the first specified event (if either of the final and inital steps take place). This example demonstrates nicely the ability of ITPSL to comply with the functional requirement FR4 (Section 4.3), as the language can clearly express a sequence of events (first, final and intermediate steps), in response to Meier's event description dimensions [36].

The very first event of the threat prediction sequence is described by the third and last ITPSL signature sub-block. On the basis of knowing that the Opera and Mozilla Firefox browsers are used, a search string of the LOIC DDoS tool website URL is specified as part of a 'fileexists' ITPSL directive. The user might use either of the two web browsers and thus the file location could either be under the user's .mozilla or .opera subdirectories, where the browsers are likely to store the web history files.

The previously mentioned three steps are not the only predictive path an insider could follow, in order to execute the LOIC DDoS tool. One could envisage different paths. For example, the user could execute the LOIC executable by copying the uncompressed file from a USB key (no web searches and no decompression steps) to his home area. Yet another alternative predictive path would be to execute steps 1, 2 and 3, but then immediately erase the web history of his web browser. This would invalidate step 1. The alternative paths should not invalidate the entire signature. This is where the decision theoretic features of the language come into the game in order to handle the uncertainty.

Figure 3.2 explained the limits between threat prediction and threat detection. A major difference between a threat prediction and a threat detection ITPSL compiler context is that the step consumption is inclusive in threat prediction. In other words, any of the steps could trigger/validate the signature. In contrast, threat detection has a more selective step consumption and will validate the signature only when all the steps occur in order ('as_a_result_of' operation).

The more relevant events one adds prior to the final threat prediction step, the more likely to increase the prediction capability of the produced signature. However, the ITPSL Weight Matrix concept (Figure 38, Section 6.4) should reward the final events with a greater weight than that of their predecessors. This is useful in case the previous predictive steps become invalid (they will not match). For this reason, the ITPSL header (Figure 7.7) had a Weight Matrix of 3,10,20,70. The final step score (70) indicates this strategy. There is no way that one can achieve the execution of a file by not placing it uncompressed and ready to run in places where file permissions allow him to. Thus, even if all the other steps are invalidated, at least the final one would give the largest possible event score.

As with Section 7.5, the signature is submitted to the ITPSL signature repository for validation and re-use:

```
[root@cn1 itpslcomp]# ./submitsig.pl scenario3.xml

...
submitsig.pl SUCCESS: Inserted signature with MD5 hash e957b4fa3f6cbc90200c339b6435e2a5 into the ITPSL
repository.
```

and subsequently fed to the ITPSL compiler by means of the following script:

```
[root@cn1 itpslcomp]# while true; do date ; ./itpslparser.pl scenario3.xml; sleep 60; done
```

This script keeps submitting the predictive signature to the ITPSL compiler every minute. The script was started towards the beginning of the IT misuse game (18/12/2011 at 01:02:20 secs), and hence the first EPMO scores were zero for the various hosts and users:

```
Tue Dec 18 01:02:20 CET 2010
proteas.uio.no:
(johnc,0)
(agrico1,0)
(agrico2,0)
(agrico3,0)
(engineer1,0)
(engineer2,0)
(engineer3,0)
dionisos.uio.no:
(johnc,0)
(agrico1,0)
(agrico2,0)
(agrico3,0)
(engineer1,0)
(engineer2,0)
(engineer3,0)
....
Tue Dec 18 01:03:20 CET 2010
proteas.uio.no:
(johnc,0)
(agrico1,0)
(agrico2,0)
(agrico3,0)
(engineer1,0)
(engineer2,0)
(engineer3,0)
...
```

However, 10 days later, the ITPSL compiler warns:

```
Tue Dec 28 11:19:32 CET 2010
proteas.uio.no:
(johnc,10)
(agrico1,0)
(agrico2,0)
(agrico3,0)
(engineer1,0)
(engineer2,0)
(engineer3,0)
…
Tue Dec 28 11:24:32 CET 2010
proteas.uio.no:
(johnc,80)
(agrico1,0)
(agrico2,0)
(agrico3,0)
(engineer1,0)
(engineer2,0)
(engineer3,0)
```

A score of 80 indicates that the compiler fired on the very final step and first step (70+10). A look at the user's home directory verifies the existence of the files in question:

```
-rw-r--r--.  1 johnc johnc  35147 Sep  8  2010 COPYING
drwxr-xr-x.  2 johnc johnc   4096 Dec 28 11:23 images
-rw-r--r--.  1 johnc johnc      1 Sep 19  2007 INSTALL
-rwxr-xr-x.  1 johnc johnc 175590 Dec 10 21:02 loiq
-rw-r--r--.  1 johnc johnc    752 Dec 10 03:28 loiq.pro
```

```
-rw-r--r--.  1 johnc johnc    196 Oct 16 21:40 loiq.qrc
-rw-r--r--.  1 johnc johnc   9647 Dec 10 03:28 Makefile
-rw-r--r--.  1 johnc johnc   1443 Oct 14 18:17 README
drwxr-xr-x.  2 johnc johnc   4096 Dec 28 11:23 src
drwxr-xr-x.  2 johnc johnc   4096 Dec 28 11:23 translations
```

In addition, the following file validates the first part:

```
[root@cn1 .opera]# fgrep loiq *
Binary file download.dat matches
global_history.dat:loiq | Download loiq software for free at SourceForge.net
global_history.dat:http://sourceforge.net/projects/loiq/
global_history.dat:Download loiq from SourceForge.net
```

The date stamps of the files indicate that the directory has been created around 11:23 hours on Dec 28th. At this point the compiler has sensed correctly the presence of the files, but found no intermediate step on decompressing the tarball. This can be attributed to the 100 ms sampling process execution sampling frequency (Section 5.3, Chapter 5). Inevitably, LUARM could miss some commands and this is another disadvantage of the audit engine that needs to be addressed.

Nevertheless, the system did manage to predict/warn the system administrator. After the alert, the system administrator could stop the user from using the program successfully. The entire MySQL data set for this scenario has been placed online for reference [82].

7.7 Detecting and preventing accidental information leakage

Accidental and intentional information leakage attributed to insiders is a serious threat for IT systems and a number of efforts have been made to address this type of IT misuse [159] [160]. Detecting and preventing this type of threat is important and demonstrated by the fourth IT misuse game scenario.

Prior getting into the details of the fourth scenario, it is worth providing a short discussion of how the case relates to common operating system information access control features. Today, most operating systems control access to information (file and process execution) by extending the concept of Discretionary Access Control (DAC) mechanisms [161]. The Orange Book [69] defines DAC as “a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)”. Assigning access permissions on a user and user group basis on computer files is a good example of a DAC.

In contrast, the same source [69] defines Mandatory Access Control (MAC) as “a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity".

A MAC based security mechanism beyond the basic access controls is commonly employed in Unix/Linux and Microsoft based operating systems. Security Enhanced Linux (SELinux) [162] and

Mandatory Integrity Control [163] are two examples of MAC mechanisms for Linux and Windows operating systems respectively.

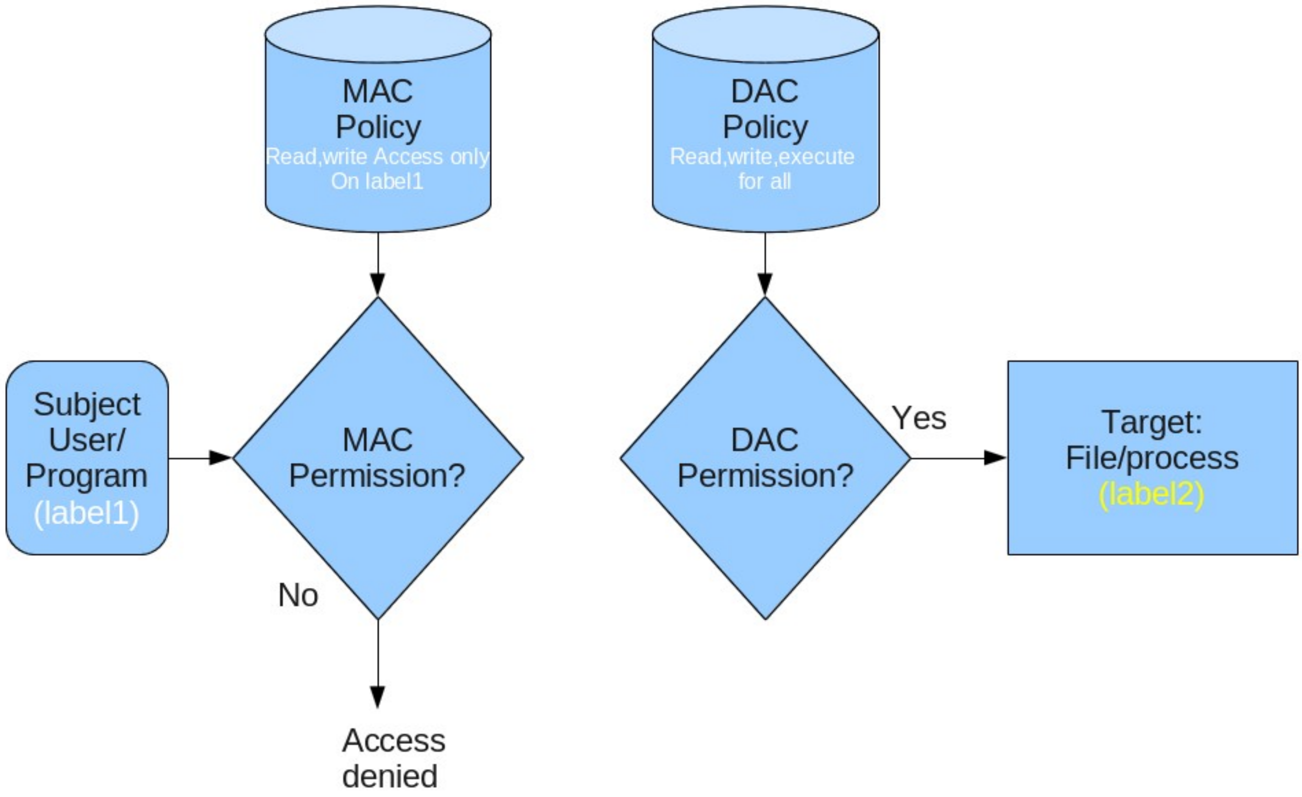
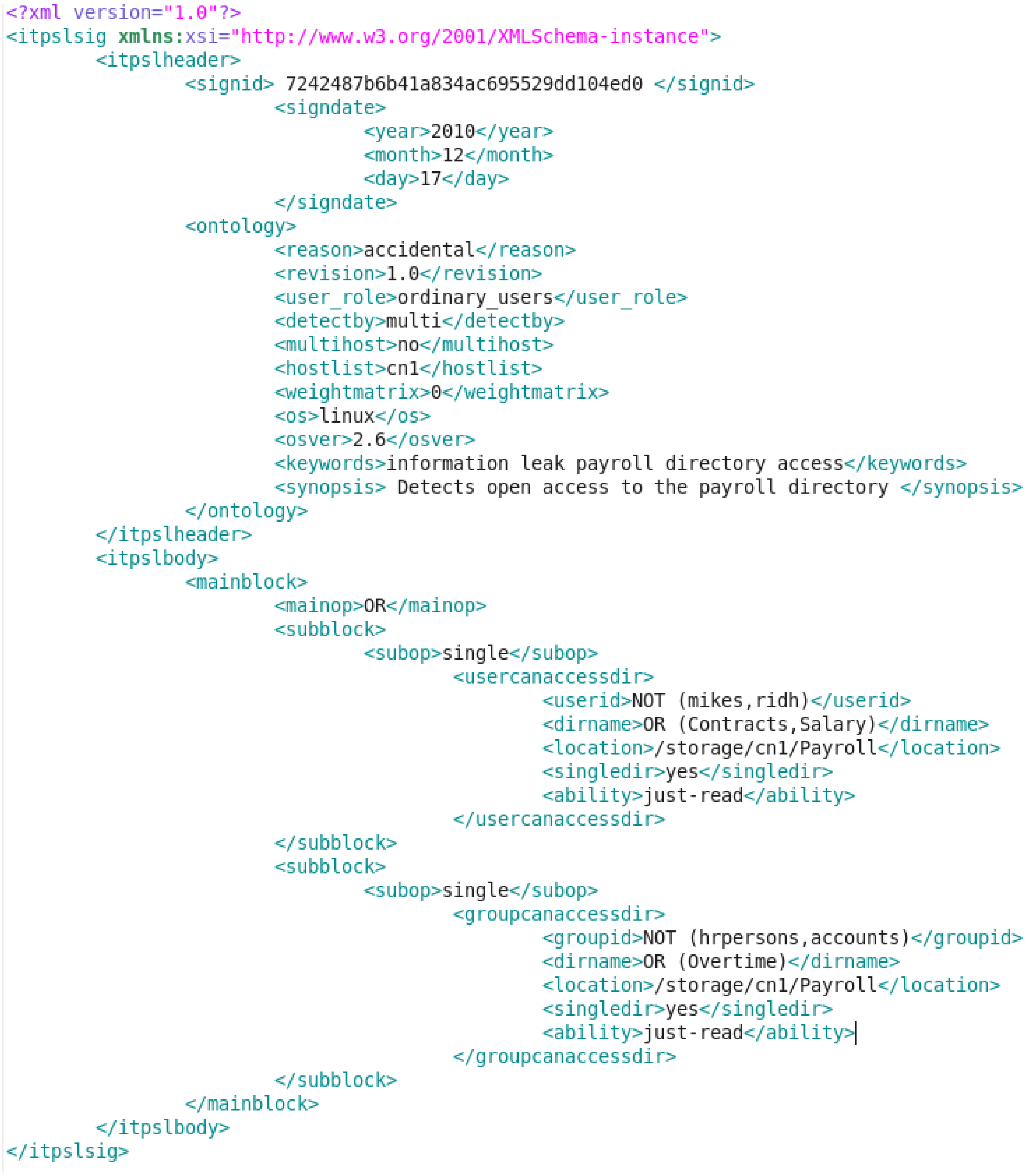


Figure 7.9: Combination of MAC and DAC mechanisms for file access control

MAC and DAC mechanisms are always combined in a modern operating system with MAC checks preceding the DAC ones. Figure 7.9 displays an example of this combination. On the left hand side, a user (or a program that runs with certain user id credentials) is requesting read and write access on a target file (right hand side). Beyond the traditional file access permissions defined by the file system (read, write and execute), the subject and target have a context label. This context label enforces the MAC policy and is allocated using different criteria, in order to further compartmentalize subjects and targets within certain logical groups. In the example of Figure 7.9, access is denied because the subject and target have different labels. Despite the DAC policy allowing access to everyone on the target, the MAC policy is executed first and hence access is denied.

Despite the presence of effective access control mechanisms, the detection and prevention of accidental information leaks will always require a check on the subject side in relation to certain target resources. In other words, irrespective of how many policies and mechanisms exist between the subject and target object, the most reliable way is to check what the subject can really access. This is the functional scope of ITPSL directives such as 'usercanaccessfile' and 'usercanaccessdir' (file access ability statements, Section 6.5.2).

Figure 7.10 displays the ITPSL signature that detects the information leak of scenario 4. This shows how access control policies could be verified by ITPSL semantics. The main-block consists of two sub-blocks. The first one contains a 'usercanaccessdir' directive as the access control policy of scenario 4 mentions that only certain user names (mikes,ridh) should access the 'Salary' and 'Contracts' folders. The <userid>NOT (mikes,ridh)</userid> tag is expanded to all other userids of the host and the directory access check is performed for each one of them. In contrast, the second sub-block contains a 'groupcanaccessdir' because the other part of the policy defines two distinct groups (hrpersons, accounts) that should have the only access to the 'Overtime' folder. Thus, the <groupid>NOT (hrpersons,accounts)</groupid> tag is expanded to all non system groups (root, wheel) and then to all resulting usernames that belong to these groups, in order for the check to be performed for each user account member.



```
<?xml version="1.0"?>
<itpslsig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <itpslheader>
                <signid> 7242487b6b41a834ac695529dd104ed0 </signid>
                        <signdate>
                                <year>2010</year>
                                <month>12</month>
                                <day>17</day>
                        </signdate>
                <ontology>
                        <reason>accidental</reason>
                        <revision>1.0</revision>
                        <user_role>ordinary_users</user_role>
                        <detectby>multi</detectby>
                        <multihost>no</multihost>
                        <hostlist>cn1</hostlist>
                        <weightmatrix>0</weightmatrix>
                        <os>linux</os>
                        <osver>2.6</osver>
                        <keywords>information leak payroll directory access</keywords>
                        <synopsis> Detects open access to the payroll directory </synopsis>
                </ontology>
        </itpslheader>
        <itpslbody>
                <mainblock>
                        <mainop>OR</mainop>
                        <subblock>
                                <subop>single</subop>
                                        <usercanaccessdir>
                                                <userid>NOT (mikes,ridh)</userid>
                                                <dirname>OR (Contracts,Salary)</dirname>
                                                <location>/storage/cn1/Payroll</location>
                                                <singledir>yes</singledir>
                                                <ability>just-read</ability>
                                        </usercanaccessdir>
                        </subblock>
                        <subblock>
                                <subop>single</subop>
                                        <groupcanaccessdir>
                                                <groupid>NOT (hrpersons,accounts)</groupid>
                                                <dirname>OR (Overtime)</dirname>
                                                <location>/storage/cn1/Payroll</location>
                                                <singledir>yes</singledir>
                                                <ability>just-read</ability>
                                        </groupcanaccessdir>
                        </subblock>
                </mainblock>
        </itpslbody>
</itpslsig>
```

Figure 7.10: ITPSL signature for Scenario 4

The signature was successfully submitted to the ITPSL repository and then the ITPSL compiler was invoked on 18/12/2010:

```
[root@cn1 itpslcomp]# while true; do date ; ./itpslparser.pl scenario3.xml; sleep 60; done
```

On the 29th of December 2010, the compiler successfully intercepted the condition which could enable an information leak:

```
Tue Dec 29 11:49:32 CET 2010
proteas.uio.no:
(johnc,1)
(agrico1,1)
(agrico2,1)
(agrico3,1)
(engineer1,1)
(engineer2,1)
(engineer3,1)
```

Each of the resulting lists above indicates that the compiler can successfully intercept a deviation from the specified access policy. This deviation was simulated by one of the game users who opened the filesystem privileges by accident to the public around 11:44 local time. This was confirmed during the debriefing of the misuse game participants.

7.8 Conclusions

This Chapter has presented the ITPSL compiler system and four indicative real world IT misuse scenarios for the purposes of evaluating the abilities of the language and its underlying audit log structure (LUARM). As the project did not obtain permission to publish the real world data from

LUARM adopters, an IT misuse game was set in order to replay the data and thus demonstrate the effectiveness of the language. The game scenarios demonstrated how an IT specialist (system administrator, data forensics officer) could combine the relational power of the LUARM audit record and the language semantics to prove beyond reasonable doubt the insider actions and even predict some insider threats.

The results indicated an overall satisfactory detection and prediction ability for the proposed language and its associated audit record for all four of the simulated game scenarios. However, some serious deficiencies were also detected. The first one concerns a difficulty of LUARM to get network endpoint DNS data, so that suspicious web pages and endpoints URLs can be reliably collected. This could potentially result in important evidence loss, especially in threat prediction context, as many threat indicators could relate to the DNS name of the user initiated endpoint.

An additional notable LUARM deficiency relates to its process execution sampling frequency. The data analysis and subsequent debriefing on scenario3 showed that a 100 ms sampling frequency is too slow to log accurately most of the user process execution steps (the decompression command seems to have been lost). The weight matrix concept compensated in that case, as the LUARM engine has intercepted the initial and final signatures threat prediction steps. However, loosing steps due to slow sampling frequency in an arbitrary manner introduces reliability problems with the collected data and this issue needs to be addressed.

Chapter 8 addresses the previously mentioned issues and provides an overall evaluation of ITPSL/LUARM in a wider context using the criteria mentioned in Section 7.1 .

Chapter 8 Conclusions

The previous chapter presented LUARM and ITPSL in action. It also set the rules for an overall evaluation of the ITPSL/LUARM framework. This chapter concludes the thesis by providing an overview of addressing some of the major achievements and tool deficiencies that rose from the game scenario. The discussion starts by taking a look at the pros and cons of the devised audit record format. An evaluation what has been achieved in terms of the ITPSL functional requirements of Section 4.3 and the ITPSL language evaluation criteria of Section 7.1 follows. Finally, this chapter concludes by considering future research directions in terms of improving ITPSL. Special consideration is given to the issue of user privacy as part of the LUARM audit record format.

8.1 LUARM achievements and deficiencies

LUARM's main achievement is the provision of an audit record format that is specifically tailored for insider misuse detection and prediction. Section 5.2 presented a sample of existing audit engines and their deficiencies in terms of misuse detection. LUARM addresses those deficiencies satisfactorily by means of:

- Providing a unified and detailed level of system activity (file, network and process execution) by combining both static and dynamic/volatile system information in the audit record.
- Giving emphasis to the provision of user accountability. The entire structure of the audit record is designed around the concept of recording user actions and linking them to the detailed log of events.
- Taking care of various data reliability issues: The collected data do not reside on the client side where they can be vulnerable to malicious alteration. They are stored by default on a central server where timing and data storage availability issues are taken care of.

After the conclusion of the game scenarios of Chapter 7, it is clear that the audit record structure is sufficient to express a range of common IT misuse scenarios. Searching through audit data in relational mode is not a novelty in itself. However, the overall combination of user accountability and system-level detail is a unique combination which provides a much better way to find evidence than shifting through dispersed text files.

Nevertheless, the game scenarios revealed two major issues with the LUARM audit record format. Both of these issues are considered important as they affect the availability of important evidence.

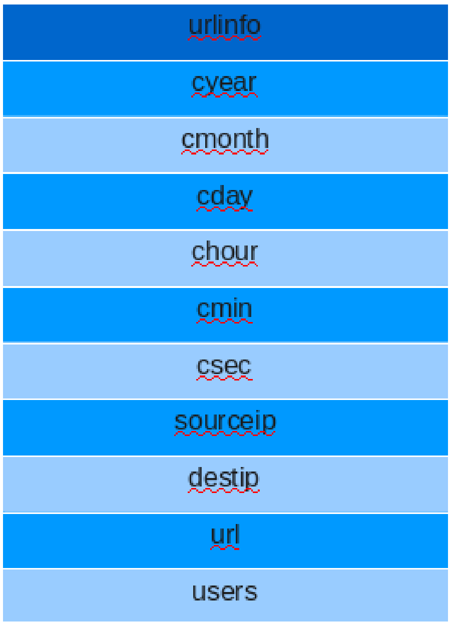
The first issue relates to the sampling frequency of user processes execution. After the debriefing of the third misuse game scenario (Section 7.6), it became evident that LUARM was loosing process execution data. A fault was located at the process execution monitoring module 'psactivity.pl' [82]. Due to the way the sampling loop was written in that module, the effective sampling frequency could exceed by far the desired 100 millisecond sampling frequency. As a result, LUARM would miss processes that executed by various users in the system.

The module was re-written using an entirely different process execution sampling philosophy. A Linux kernel technique called 'execve wrapping' was employed by adopting the Snoopy logger open source software [164]. The 'execve' system call [85] is used to launch processes in the linux kernel. A modified 'execve wrapper' logger like [164] provides a way to log the process execution and its arguments without relying on a sampling loop and is thus a more efficient interface to capture live process execution data. This solved the problem of loosing process execution data due to a slow sampling rate and thus corrected an important deficiency of LUARM and ensures that process data collection is done in a reliable manner.

The pornography detection scenario (Section 7.5) revealed another important deficiency of the LUARM audit record: The inability to reverse DNS resolve certain IP addresses. This is a consequence of two important factors:

- The core design decision of LUARM to avoid DNS name resolution on the client for reliability purposes, as described in Section 5.3.
- The fact that many adult websites do not have reverse DNS records configured properly. This is probably an unintentional administrative error or an intentional omission, as the Internet hosting providers of these websites might also engage in activities such as email spam operations and even rogue websites for 'phising' attack [165] purposes. As a result, the technical community has proposed the technique of Forward-Confirmed reverse DNS (FCrDNS) records [98] where a given IP address has forward (name-to-address) and reverse (address-to-name) DNS entries that match each other consistently.

The end result is that LUARM might loose a valuable piece of evidence (the URL pointed by the browser). The situation was remedied by employing the 'urlsnarf' utility[166] which runs on the client side ('urlcollector.pl' module). The end result is that LUARM logs all the URLs that are visited by the client system and enters them in the LUARM 'urlinfo' database table (Figure 8.1). The 'url' column logs the actual URL visited by the client system. The collected URL information can then be easily cross-referenced to the information of the 'netinfo' table (Figure 5.10). A correlation of the 'destip', 'users' and timing information fields with the relevant fields of the 'netinfo' table can link the logged URLs to specific user actions and applications. Hence, LUARM can now log a useful piece of evidence. Even if the client DNS configuration is unreliable, this can be flagged by comparing the client resolved URL to the DNS resolution of the server side. If the two differ, then the client URL information can be discarded.



urlinfo
cyear
cmonth
cday
chour
cmin
csec
sourceip
destip
url
users

Figure 8.1: The 'urlinfo' LUARM table

The resolution of these two issues makes LUARM a more reliable evidence collection engine. The next section presents a critique of the effectiveness of the language semantics.

8.2 ITPSL achievements and deficiencies

LUARM is the data repository for insider misuse detection and prediction. ITPSL is the mechanism which shifts through the data in search of misuse incidents or signs of them (misuse prediction). Section 4.3 provided a list of functional requirements (FRs) for the proposed language. A combination of these requirements and the results of the game scenarios reveals that ITPSL has already met most its functional objectives. In particular:

- Its underlying audit record architecture stores the data away from vulnerable client systems (FR1).
- The submitsig.pl (Figure 7.2) and searchsig.pl (Figure 7.3) ITPSL utilities can create insider misuse incident signature repositories, as dictated by requirement FR2.
- In response to requirement FR3, ITPSL does combine the description of static and live forensic data under one common semantic framework.
- ITPSL is the first misuse detection language to support decision theoretic information (FR5) with its weight matrix concept (Figure 6.8). The association of weights and events facilitates insider threat prediction based on the analyst's view of how the threat precursors occur at file, network, process execution and hardware device level.
- The block-based declarative structure of events (Figure 6.11) provides a hierarchical way of describing events (FR6) which can specify a precise order of events (FR4).
- Despite the fact that none of the replayed scenarios provided a true opportunity to test event correlation across multiple hosts, ITPSL is equipped with operators such as the <onhost> tag (Section 6.4), which could bind certain events to specific hosts and thus permit cross-host event correlation.

However, there are certain areas where ITPSL has weaknesses, despite the detection/prediction success of the benchmark game scenarios.

One important issue to consider is the expressive granularity of the events. ITPSL (and its underlying audit record) capture file, network endpoint and process creation data. All applications generate these types of events in an operating system. However, not all of these events are meaningful to insider misuse detection and prediction at that level. There is a category of applications whose file, network and process execution operations are not easy to interpret. A great example is that of database applications where the observation of file access patterns do not reveal any clues about what kind of information is accessed and in what manner. In order to address that problem, the IDS/IPS community suggests application-integrated monitoring [167], a technique where audit records are generated by monitoring routines internal to the application itself. These routines know which internal events are of interest and can export relevant activities in specific audit log formats.

ITPSL (and its underlying logging mechanism) clearly needs certain semantic extensions to monitor applications such as databases, social networking sites and other applications that organize information using internal mechanisms. Everything else can be described by the proposed file, network, process and hardware semantics.

A last but equally important issue to consider is that of the monitoring framework scalability. LUARM is a research prototype effort. Hence, it's code is not optimized for real world large IT infrastructure deployment. Code optimization is not within the academic discipline of this research project. Nevertheless, the LUARM concept enforces a relational model [66] and the SQL interface

[84]. This was a core design choice aiming to enhance the correlation versatility of the audit log structure. This goal was achieved, however, it imposes certain scalability limitations.

Relational databases have been at the forefront of massive data storage and organization for several decades. A number of different approaches enable Relational Databases to scale well, so that they can handle concurrently a large number of operations (often referred to as “Transactions Per Second – TPS benchmark”) [168]. One common technique is the partitioning of tables amongst various computational nodes, as well as database clustering, a technique where database operations are load balanced in dedicated RDBMS modes [169]. Both of these techniques have been employed successfully to handle millions of concurrent transactions per second in environments such as telecommunication processing and High Performance Computing.

Despite the success of employing Relational Database scalability measures, every practitioner agrees that the measures can become a cumbersome process and they have limits [169]. As the Relational Database scales horizontally (spread in various nodes), the complexity of managing software and hardware aspects in relation to interprocess communication increases. In addition, large data processing problems that require several hundred million transactions per second (examples are social networking site activities, large Petaflop scale simulations) cannot be accommodated by Relational Databases.

The previous complexity and transaction volume requirements created a new generation of database products that are collectively referred to as 'NoSQL databases' [170]. The 'NoSQL' term emphasizes the departure of these products from the traditional relational model [66], in an attempt to balance the need to scale and the need to preserve some of the properties of relational consistency [66].

These products deviate from traditional RDBMS requirements such as data normalization and create simpler but faster key lookup mechanisms, in order to achieve massive concurrency and scalability. The term 'eventually consistent' coined by Werner Vogels [171] embodies all these principles.

LUARM's SQL table schema (Chapter 5) is simple and does not require data normalization or embody any relational key constraints amongst the various client tables (audit levels). Consequently, it should be possible to port the audit log structure into a 'NoSQL' product and take advantage of its speed and scalability features, should the size of the monitored infrastructure makes the employment of a Relational Database product prohibitive.

8.3 Future directions of insider IT misuse detection and prediction

The basic goal of the research project was to produce a prototype system that can systematize the specification of insider threats and their precursors, as well as establishing signature repositories that can aid researchers build best recipes for detecting and predicting insider IT misuse. The goal was achieved to a large extent, but there are often huge gaps between the research laboratory and the production use of a concept. Beyond the deficiencies mentioned in the previous sections of the chapter, these gaps define the ways the research work will evolve from a concept idea to a usable system.

In the Information Security world, there is always a tension between monitoring and user privacy, as most systems need to retain and collect data about a user's on-line behavior. In direct contrast, privacy dictates the right of individuals to define whether somebody will collect data about their on-line actions and the extent or way the data can be used [172]. Insider IT misuse monitoring is no exception. The LUARM audit trails and the ITPSL signatures that shift through the collected audit data inevitably identify personal aspects of system users and breach certain aspects of user privacy. Addressing these privacy issues is a complex task, driven by legal compliance and an ethical basis.

In most European countries, personal data is protected by privacy law [173], which permits the individual to determine himself issues related to the disclosure and use of his personal information. As it is impossible to participate in a society without disclosing some level of personal information, most privacy laws grant exceptions from that personal right. These exceptions provide the legal and ethical ground for audit systems to collect personal information.

The same laws dictate that a control of the amount and type of logged data must be performed. This can be achieved by pseudo-anonymizing certain parts of the audit record, in order to protect certain aspects of the user privacy but still be able to infer IT misuse reliably. The term 'Privacy-Respecting Intrusion Detection' [146] encompasses all the efforts of achieving a good compromise between the need to monitor and the need to respect user privacy. In addition, a trust-authorization schema needs to be established between three entities (the term 'entity' might refer to a combination of human and software module elements of a procedure):

- the entity who will indicate a legitimate need to access the audit logs (access request)
- the entity who will reverse the pseudo-anonymization of audit log data and recover the information (log reader)
- the entity who will infer the IT misuse (analyst)

As an example, on the suspicion of an IT misuse incident in a firm, the access request entity might be the IT officer of the firm, the log reader entity may come from an externally assigned audit company, whereas the analyst might be a team of people from the original firm and the audit company. The independence of the discrete entities (IT officer, external auditor) is proportional to the user privacy awareness of audit log processing. However, this distributed audit log access scheme requires a framework of authorizations, in order to enable trusted mediation amongst the related entities. Research on Public Key Infrastructure (PKI) [174, 175] applied to the subject of secure mediation [176] indicate the magnitude and complexity of the problem. As a result, certain parts of the LUARM audit record need to be re-designed so that they protect the privacy of the monitored users, in order to prevent illegal and unauthorized correlation of personal information.

Another important research direction for misuse detection and prediction research is how best to enablethe LUARM/ITPSL framework in virtual operating system environments [68]. In a world where operating systems are run in virtual mode for improving the efficiency of data centers and enabling Cloud Computing architectures [177], one of the main challenges is to improve the efficiency and security of the data audit components.

In their current form, LUARM and ITPSL rely on operating system components to derive the file, network endpoint, process execution and hardware device attachment information. This might be sufficient for the construction of a prototype system in the research laboratory, but it might cause a number of problems in a production system that runs hundreds or thousands of virtual operating instances:

- The cumulative CPU load of running the LUARM clients on each virtual operating system might create resource contention issues.
- The operating system modules might be often compromised or vulnerable by various types of threats and thus provide incorrect information. The variability and size of the operating system code base might make the fixing and detection of these threats difficult.

For these reasons, preliminary research [178] suggests the moving of the client audit components to the 'hypervisor', the software module that is responsible for coordinating the execution of all guest operating systems in a virtualized environment. The advantages of moving the audit sensors to the hypervisor address exactly the previous two production issues: It is more efficient to probe for the information inside the virtual machine management logic. Moreover, as the hypervisor's code base is substantially smaller than that of a full operating system, it is easier to manage security risks related to code flaws and produce reliable components that provide audit information.

The trend to reduce the size of the code base as a measure of reducing the exposure to malware attacks is evident and in more recent research approaches. The Strongly Isolated Computing Environment (SICE) [179] employs aspects of hardware and software approaches and a code base size of just 300 lines of code, in order to assure the integrity of computational tasks in modern commodity multi-core processors.

Adapting LUARM to utilize research approaches like the ones of [178] and [179] will defend it against hypervisor and guest operating system vulnerabilities and provide a reliable way of system activity introspection.

8.4 The contribution of this research project

This thesis has demonstrated the seriousness of the insider threat. It is a real problem that affects many organizations and is here to stay, as no current solution represents a panacea for tackling the problem and our dependency on computing infrastructures is increasing.

A large part of the difficulty of tackling insider IT misuse comes from the fact that there are fundamental deficiencies in defining both what constitutes misuse is in an organizational context, as well as how IT misuse manifests itself at system level. The proposed ITPSL framework (semantics plus logging) addresses exactly this lack of systemic definition. Providing a basis for insider threat specification is an important step in the fight of dealing with these types of threats.

A second but equally important ability to address insider threats (and the deficiencies of previous intrusion specification languages) concerns the ability to predict threats. The ITPSL weight matrix concept provides a basic mechanism that associates a systemic description of the threat precursor to a numeric confidence indicator. This arms the IT security specialist with a simple tool to describe consistently what comes before the actual misuse act, a feature that does not exist in generic intrusion specification languages.

Insider threat specification is of course one component of the evolving and multi-factorial IT misuse world, but an important one. Human management and legal issues relating to what constitutes IT misuse and how can IT misuse be monitored and proven in a court of law perplex the problem even further and require different (non technical) knowledge in order to be fully addressed.

However, all of these non technical issues relate to the way threats and misuse actions are expressed and defined. In a world of increasing and evolving user interactions and pervasive computing paradigms insider threat specification is an important stone in the wall of combating harmful user actions. This project has provided such a stone by drawing a theoretical foundation for insider threat specification with prototype tools that monitor the IT infrastructure and aid the researcher/practitioner to express and use observable events as threat indicators at the IT infrastructure level.

References

1. Pfleeger C., Pfleeger S. (2003), ‘Security in Computing’, Third Edition, Prentice Hall, ISBN:0130355488: Page 6 contains the definition of the term “threat” in an information security context, ppp. 198-200 and 451 explain the concept of the Access Control List (ACL).

2. PHYS.ORG web portal (2009), “Hackers breach US air traffic control computers”, Article published on 8th May 2009, http://www.physorg.com/news161028232.html (Last Accessed 02/04/2011)

3. Appel A.W., Ginsburg M., Hurstli H., Kernigham B.W., Richards C.D., Gang T. (2008), “Insecurities and Inaccuracies of the Sequoia AVC Advantage 9.00H DRE Voting Machine ”, Technical Report, Princeton University. http://coblitz.codeen.org/citp.princeton.edu/voting/advantage/advantage-insecurities-redacted.pdf (Last Accessed 02/04/2011)

4. Furnell S. (2001), “Cybercrime Vandalizing the Information Society”, Addison-Wesley Professional, ISBN: 978-0201721591.

5. Erickson J. (2008) “Hacking: The Art of Exploitation, 2nd Edition”, No Starch Press, ISBN: 978-1593271442.

6. Richardson R. (2010), “15th Annual 2010/2011 Computer Crime And Security Survey”, http://gocsi.com/2010_survey_purchase (Last Accessed 02/04/2011)

7. InforSecurity Europe and PwC (2010), “INFORMATION SECURITY BREACHES SURVEY 2010 | technical report”,http://www.pwc.co.uk/eng/publications/isbs_survey_2010.html (Last Accessed 02/04/2011)

8. Furnell S. (2004), “Enemies within: the problem of insider attacks”, Computer Fraud and Security, Volume 2004 Issue 7, pp. 6-11.

9. Magklaras, G. (2005), An Architecture for Insider Misuse Threat Prediction in IT Systems, MPhil Thesis, School of Computing, Communications and Electronics, University of Plymouth, UK.

10. Caelli, W., Longley, D. and Shain, M. (1991), Information Security Handbook, Stockton Press.

11. Probst C., Hunker J., Bishop M., Gollman D. (2009), “Countering Insider Threats”, ENISA Quarterly Review Vol. 5, No. 2, June 2009, pp. 13-14.

12. Computer Security Institute (2001), “2001 CSI/FBI Computer Crime and Security Survey”, Computer Security Issues & Trends, Vol. VII, NO.1. In addition, the following readers should refer to the following technical discussion on the Slashdot web portal which reveals details of Hansen's data hiding techniques:Slashdot (2001), “Spying and Technology: Robert Philip Hanssen”, posting on Slashdot.org, 22 February 2001, http://slashdot.org/articles/01/02/22/0622249.shtml (Last Accessed 03/03/2011)

13. Computerworld Internet Portal (2000), “The Cyber-Mod Squad Sets Out After Crackers”, Article written by Deborah Radcliff documenting the case of Abdelkader Smires, http://www.computerworld.com/news/2000/story/0,11280,45927,00.html (Last Accessed 03/03/2011)

14. Garfinkel S, Spafford G. (1996), ‘Practical UNIX and Internet Security’, Second Edition, O’Reilly and Associates, Sebastopol, CA, ISBN: 1-56592-148-1.

15. ZDnet Internet Portal (2001), “Firms shop around for Net law jurisdictions”, article written by Wendy McAuliffe, http://news.zdnet.co.uk/business/0,39020645,2085983,00.htm (Last Accessed 03/03/2011)

16. Cha A.(2008), “Even spies embrace China's free market.”, Washington Post, February15,2008,http://www.washingtonpost.com/wpdyn/content/article/2008/02/14/AR2008021403550.html (Last Accessed 03/03/2011)

17. Feiertag R., Kahn C., Porras P., Schnackenberg D., Staniford-Chen S., Tung B. (1999), “A Common Intrusion Specification Language (CISL)”, June 1999 revision, URL: http://gost.isi.edu/cidf/drafts/language.txt (Last Accessed 03/03/2011)

18. Spinellis D., Gritzalis D. (2002), “Panoptis: Intrusion detection using a domain-specific language.”, Journal of Computer Security, Volume 10, pp. 159–176.

19. Doyle J. (1999), "Some representational limitations of the Common Intrusion Specification Language", Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1999 Revision.

20. Amoroso, E. (1999), "Intrusion Detection: An introduction to Internet surveillance, correlation, traps, trace-back, and response", Second Edition, Intrusion.Net books, ISBN:0-9666700-7-8.

21. The Common Intrusion Detection Framework (CIDF), http://gost.isi.edu/cidf/ (Last Accessed 03/03/2011)

22. Curry D., Debar H., Feinstein B. (2004), "The Intrusion Detection Message Exchange Format", Internet Draft, Intrusion Detection Exchange Format working group, Internet Engineering Task Force, http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-11.txt (Last Accessed 03/03/2011)

23. Feinstein B., Matthews G., White J. (2002), "The Intrusion Detection Exchange Protocol Internet Draft (IDXP)", Intrusion Detection Exchange Format working group, Internet Engineering Task Force, http://www.ietf.org/internet-drafts/draft-ietf-idwg-beep-idxp-07.txt (Last Accessed 03/03/2011)

24. The W3C Extensible Markup Language web page, http://www.w3.org/XML/ (Last Accessed 03/03/2011)

25. Bace R. (2000), “Intrusion Detection”, First Edition, Macmillan Technical Publishing, Indianapolis, USA: pp. 38-39 discuss the terms 'misuse detection' and 'anomaly detection' in an intrusion specification context, pp. 47-66 discuss various audit record issues. Page 227 discusses the distinction between the security and monitoring policies. Finally, page 156 presents the 'clock skew' problem, in relation to scaling an IDS infrastructure.

26. Porras P. (1992), “A state transition analysis tool for Intrusion Detection”, University of California, paper found at http://www.csl.sri.com

27. Hopcroft J., Ullman J. (1979), “Introduction to Automata Theory, Languages, and Computation”,First Edition, Addison-Wesley. ISBN 0-201-02988-X.

28. Jensen, K. (1997), “Coloured Petri Nets”, Springer Verlag, ISBN 3-540-62867-3.

29. Lindqvist U., Porras P. (1999), “Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)”, In Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, California, May 9–12, 1999.

30. Mounji A. (1997), “Languages and Tools for Rule-Based Distributed Intrusion Detection”, PhD Thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1997.

31. Schmerl S., Koenig H., Flegel U., Meier M. (2006), “Simplifying signature engineering by Re-use”, G. Müller (Ed.): ETRICS 2006, LNCS 3995, pp. 436 – 450.

32. Eckmann S. T., Vigna G., Kemmerer R. A. (2002), "STATL: an attack language for state-based intrusion detection", Journal of Computer Security, Vol. 10, Issue 1-2, IOS Press Amsterdam, The Netherlands, pp. 71-103.

33. Pouzol J. M., Ducassé M. (2001), "From declarative signatures to misuse IDS", Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection (RAID 2001), no. 2212 in LNCS, Davis, California, USA, October 2001, Springer, pp. 1-25.

34. Kumar S., Spafford E.H. (1995), "A Software Architecture to support Misuse Intrusion Detection. Technical Report CSD-TR-95-009, Department of Computer Sciences, Purdue University, March 1995.

35. Keith D. (1993), "The Joy of Sets", 2nd Edition, Springer Verlag, ISBN 0-387-94094-4.

36. Meier M. (2004). "A Model for the Semantics of Attack Signatures in Misuse Detection Systems", K. Zhang and Y. Zheng (Eds.): ISC 2004, LNCS 3225, Springer-Verlag Berlin Heidelberg 2004, pp. 158–169.

37. Magklaras G., Furnell S., Brooke P. (2006), "Towards an Insider Threat Prediction Specification Language", Information Management & Computer Security, vol. 14, no. 4, pp. 361-381.

38. Somerville I. (2000), "Software Engineering", Addison Wesley, ISBN: 020139815X.

39. Furnell S., Magklaras G., Papadaki M., Dowland P. (2001), ‘A Generic Taxonomy for Intrusion Specification and Response’, Proceedings of Euromedia 2001, Valencia, Spain, pp. 125-131.

40. Tuglular T. (2000), “A preliminary Structural Approach to Insider Computer Misuse Incidents’, EICAR 2000 Best Paper Proceedings, pp. 105-125.

41. Aslam T., Krsul I., Spafford E.(1996), “Use of a Taxonomy of Security Faults”, Technical Report TR-96-051, COAST Laboratory, Department of Computer Sciences, Purdue University, IN, 1996.

42. Bach M. (1986), “The design of the UNIX Operating System”, Prentice Hall International Editions, NJ, 1986. Page 10 defines the term “process” in an operating system context, ISBN: 0-13-201757-1.

43. Richter J. (1997), “Advanced Windows”, Microsoft Press, Redmond, Washington.

44. Magklaras G., Furnell S. (2004). “The insider misuse threat survey: investigating IT misuse from legitimate users”m Proceedings of the 5th Australian Information Warfare & Security Conference, Perth Western Australia, 25-26 November 2004, pp. 42-51.

45. Papadaki M., Magklaras G., Furnel S., Alayed A. (2001), “Security vulnerabilities and System Intrusions – The need for Automatic Response Frameworks”, Proceedings of the 2001 Small Systems Security International Conference, Las Vegas, USA, September 2001.

46. Moore, D., Voelker, G. and Savage S. (2001), "Inferring Internet Denial of Service Activity", Proceedings of the 10th USENIX Security Symposium, Washington D.C, August 2001, pp. 9-22.

47. Frykholm, N. (2000), "Countermeasures against Buffer Overflow Attacks", White Paper, RSA Laboratories.

48. Postel, J. and Reynolds, J. (1983), "TELNET Protocol Specification", Request For Comments (RFC) 854, IETF Network Working Group, May 1983.

49. Ylonen, T. (1995), The SSH (Secure Shell) Remote Login Protocol, Internet Draft, IETF Network Working Group, 15 November 1995, http://www.free.lp.se/fish/rfc.txt (Last Accessed 18/05/2011)

50. Ziegler, R. (2002), LINUX Firewalls, Second Edition, New Riders Publishing, Chapter 2, pp. 48-50.

51. Sharda, N. (1999), Multimedia Information Networking, Prentice Hall Inc., Chapter 12.

52. Wood B. (2000). "An insider threat Model for Adversary Simulation", SRI International, Research on Mitigating the Insider Threat to Information Systems - #2: Proceedings of a Workshop Held by RAND, August 2000.

53. Schultz, E.E. (2002). “A framework for understanding and predicting insider attacks”, Computers & Security, vol. 21, no. 6, pp. 526-531.

54. Shaw E.D., Ruby K.G., Post J.M. (1998), “The Insider Threat to Information Systems”, Security Awareness Bulletin, No. 2/98, Political Psychology Associates Ltd.

55. Brancik K.C. (2008), ”Insider Computer Fraud An in-depth Framework for Detecting and Defending Against Insider IT Attacks”, Auerbach Publications, Taylor & Francis Group, ISBN 1-4200-4659-4.

56. Magklaras G., Furnell S. (2005), “A Preliminary Model of End User Sophistication for Insider Threat Prediction in IT Systems”, Computers & Security, Volume 24, Issue 5, August 2005, pp. 371-380.

57. NSTISSAM. (1999), “The Insider Threat To US Government Information Systems”, NSTISSAM INFOSEC /1-99, U.S. National Security Telecommunications And Information Systems Security Committee, http://www.cnss.gov/Assets/pdf/nstissam_infosec_1-99.pdf (Last Accessed 18/05/2011)

58. Cappelli D., Moore A., Shimeall T.J., Trzeciak R. (2006). “Common Sense Guide to Prevention and Detection of Insider Threats”, 2nd Edition Version 2.1, Carnegie Mellon University Cylab,http://www.cert.org/archive/pdf/CommonSenseInsiderThreatsV2.1-1-070118.pdf (Last Accessed 03/03/2011)

59. AccessData Inc. web portal, URL: http://www.accessdata.com/

60. Guidance Software Inc. web portal,http://www.guidancesoftware.com/computer-forensics-ediscovery-software-digital-evidence.htm

61. Hay B., Nance K., Bishop M. (2009), “Live Analysis Progress and Challenges”, IEEE Security & Privacy, Volume 7, Number 2, pp. 30-37.

62. Adelstein F. (2006), “Live Forensics: Diagnosing Your System without Killing it First”, Comm. ACM, vol.49, no.2, 2006, pp. 63-66.

63. The Computer Online Forensic Evidence Extractor (COFEE), Microsoft Corp, http://www.microsoft.com/industry/government/solutions/cofee/default.aspx (Last Accessed 18/05/2011)

64. Petroni N., Walters A., Fraser T., and Arbaugh W.(2006), "FATKit: A Framework for the Extraction and Analysis of Digital Forensic Data from Volatile System Memory" ,Digital Investigation Journal 3(4), pp. 197-210.

65. Carrier B. (2006), “Risks of Live Digital Forensic Analysis”, Comm. ACM, vol. 49, no. 2, 2006, pp. 56-61.

66. Codd E. (1990), “The Relational Model for Database Management”, Addison-Wesley Publishing Company, 1990, ISBN 0-201-14192-2.

67. Mills D., Delaware U., Martin J., Burbank J., Kasch W. (2010), “Network Time Protocol Version 4: Protocol and Algorithms Specification”, Internet Engineering Task Force (IETF) Request For Comment (RFC) 5905, June 2010.

68. Matthews J., Dow E., Deshane T., Wenjin H., Bongio J., Wilbur P. ,Johnson B. (2008), “Running Xen, A Hands-On Guide to the Art of Virtualization”, Prentice Hall Pearson Education, ISBN-13: 978-0-13-234966-6: Chapter 1 (pp. 1-26) is a comprehensive introduction to the concept of virtualization.

69. National Computer Security Center. (1985), “Department of Defense Trusted Computer System Evaluation Criteria”, Orange Book, DOD 5200.28-std, December 1985.

70. The Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 3, July 2009. Part 2: Functional security components, http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R3.pdf (Last Accessed 18/05/2011)

71. Jackson W. (2007),“Under Attack: Common Criteria has loads of critics, but is it getting a bum rap”, Government Computer News, date: 10/08/2007, http://gcn.com/articles/2007/08/10/under-attack.aspx (Last Accessed 18/05/2011)

72. Oracle Corporation (2010), “System Administration Guide:Security Services”, Solaris 10 Operating System, Part No: 816–4557–19 , September 2010, pp. 559-672, http://dlc.sun.com/pdf/816-4557/816-4557.pdf (Last Accessed 03/03/2011)

73. The Trusted BSD Project web portal, “OpenBSM: Open Source Basic Security Module (BSM) Audit Implementation”, http://www.trustedbsd.org/openbsm.html

74. Utilities for monitoring process actitities, http://linux.maruhn.com/sec/psacct.html

75. Gerhards R. (2009), “The Syslog Protocol”, Internet Engineering Task Force (IETF), Request for Comment (RFC) 5424, March 2009.

76. Monitorware's WinSyslog website, http://www.winsyslog.com/en/product/

77. Bishop M. (1995), “A Standard Audit Trail Format (1995)”, In Proceedings of the 1995 National Information Systems Security Conference, pp. 136-145.

78. American National Standard for Information Systems — Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986, American National Standards Institute, Inc., March 26, 1986.

79. The Unicode Standard, Version 5.0, Fifth Edition, The Unicode Consortium, Addison-Wesley Professional, 27 October 2006. ISBN 0-321-48091-0.

80. Larson U., Lundin-Barse E., Jonsson E. (2005), “METAL, A Tool for Extracting Attack Manifestations”, Detection of Intrusions and Malware, and Vulnerability Assessment, Second International Conference, DIMVA 2005, ISBN/ISSN: 3-540-26613-5, pp. 85-102.

81. “psacct process accounting misses some commands”, HP IT Resource Center forums, http://forums11.itrc.hp.com/service/forums/questionanswer.doadmit=109447626+1286381845785+28353475&threadId=1413576

82. LUARM web portal at Sourceforge, http://luarm.sourceforge.net/

83. Connoly T., Begg C. (2004), “Database Systems: A Practical Approach to Design, Implementation and Management”, Fourth Edition, International Series in Computer Science, Addison-Wesley, ISBN-13: 978-0321210258.

84. Information Technology-- Database Languages – SQL: ISO/IEC 9075(1-4,9-11,13,14):2008, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45498 (Last Accessed 18/05/2011)

85. Kerrisk M. (2010), “The Linux Programming Interface”, No Starch Press, ISBN: 978-1-59327-220-3: Chapter 23 (pp. 479-512) discusses low-level details of the Linux timer implementation. Chapter 24 (pp. 513-530) discusses the 'execve' system call.

86. Microsoft Corporation (2010), “Timers, Timer Resolution, and Development of Efficient Code”.http://download.microsoft.com/download/3/0/2/3027D574-C433-412A-A8B6-5E0A75D5B237/Timer-Resolution.docx (Last Accessed 03/03/2011)

87. The Perl High Resolution Timer module at the Comprehensive Perl Archive Network (CPAN), http://search.cpan.org/~jhi/Time-HiRes-1.9721/HiRes.pm

88. Oracle MySQL RDBMS web portal, http://www.mysql.com/

89. The Perl Programming Language web portal, http://www.perl.org/

90. The Perl Database Interface (DBI) module at the Comprehensive Perl Archive Network (CPAN), http://search.cpan.org/~timb/DBI-1.615/DBI.pm

91. Rivest R. (1992), "The MD5 Message-Digest algorithm", Internet Engineering Task Force (IETF) Request For Comment (RFC) 1321, April 1992.

92. Mendel F., Rechberger C., Schläffer M. (2009), "MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners", ADVANCES IN CRYPTOLOGY – ASIACRYPT 2009, Lecture Notes in Computer Science, 2009, Volume 5912/2009, DOI: 10.1007/978-3-642-10366-7_9, pp. 144-161.

93. Xie T., Feng D. (2009), "How To Find Weak Input Differences For MD5 Collision Attacks", International Association for Cryptologic Research (IACR) ePrint archive. http://eprint.iacr.org/2009/223.pdf (Last Accessed 18/05/2011)

94. Pogue C., Altheide C., Haverkos T. (2008), "Unix and Linux Forensic Analysis DVD Tool-kit", Syngress, ISBN: 978-1-59749-269-0.

95. Russinovich M. (2009), “PsTools”, Windows Sysinternals, Microsoft Technet portal, http://technet.microsoft.com/en-us/sysinternals/bb896649.aspx (Last Accessed 18/05/2011)

96. Socolofsky T. (1991), “A TCP/IP Tutorial”, Internet Engineering Task Force (IETF) Request For Comment (RFC) 1180 (Informational), January 1991.

97. Mockapetris P. (1987), “Domain Names – Implementation and Specification”, Internet Engineering Task Force (IETF) Request For Comment (RFC) 1035, November 1987.

98. Barr D. (1996), “Common DNS Operational and Configuration Errors”, Internet Engineering Task Force (IETF) Request For Comment (RFC) 1537, February 1996.

99. Bauer M. (2003), “Building secure servers with Linux”, O'Reilly & Associates, ISBN: 0-596-00217-3: Chapter 6, pp. 154-196.

100. Schneier B. (2001), “Managed Security Monitoring: Network Security for the 21st Century”, Computers and Security, Elsevier Science Ltd., Volume 20, pp. 491-503.

101. The Virtual Private Network Consortium (VPNC) web portal, http://www.vpnc.org/vpn-standards.html

102. Mueller S. (2006), “Upgrading And Repairing PCs”, 17th Edition, Que Publishing, ISBN: 0-7897-3404-4: Chapter 4 describes the PCI and PCI-Express buses, pp. 372-378. Chapter 15 describes the USB bus, pp. 980-989.

103.The RLIKE String Regular Expression Operator, MySQL 5.1 Manual, Oracle Corporation, http://dev.mysql.com/doc/refman/5.1/en/regexp.html (Last Accessed 18/05/2011)

104.Friedl J. (2002), “Mastering Regular Expressions”, O'Reilly, ISBN 0-596-00289-0.

105.Spinellis D. (2001), “Notable design patterns for domain-specific languages”, The Journal of Systems and Software Volume 56, Issue 1 (2001), pp. 91-99.

106.Consel, C. (2004), “From A Program Family To A Domain-Specific Language”, in Lengauer, C.; Batory, D.; Consel, C.; Odersky, M. (Eds.), Domain-Specific Program Generation, LNCS 3016, Springer-Verlag, pp. 19-29.

107.Raymond E.S. (2003) “The Art of UNIX Programming”, 1st Edition, Addison-Wesley Professional.

108.Fowler M. (2005). “Language Workbenches: The Killer-App for Domain Specific Languages?”,http://martinfowler.com/articles/languageWorkbench.html#ProsAndConsOfLanguageOrientedProgramming (Last Accessed 18/05/2011)

109.The association of LISP users portal, URL: http://www.lisp.org/alu/home

110.The Smalltalk Programming Language Portal, URL: http://www.smalltalk.org/main/

111. The Ruby Programming Language portal, URL: http://www.ruby-lang.org/en/

112.Perl PGE wikipedia reference: http://en.wikipedia.org/wiki/Parser_Grammar_Engine

113.Ousterhout J. (1998). “Scripting: Higher Level Programming for the 21st Century”, IEEE Computer magazine, March 1998 issue, pp. 23-30.

114.Ogden K. C., Richards I. A. (1989), “The meaning of meaning: A study of the Influence of Language upon Thought and of the Science of Symbolism”, Houghton Mifflin Harcourt, re-issue edition (1923), ISBN: 978-0156584463.

115.Chandler D. (2007), “Semiotics: The Basics”, Second Edition, Routledge, ISBN: 978-0415363754.

116.Kleppe A. (2009), “Software Language Engineering – Creating Domain Specific Languages Using Metamodels”, Addison-Wesley/Pearson Education, ISBN: 978-0321553454.

117.Schmidt D. (1986), “Denotational Semantics: A Methodology for Language Development”, Allyn and Bacon, Boston.

118.Schach S. (2006), “Object-Oriented and Classical Software Engineering”, Seventh Edition, McGraw-Hill, ISBN: 0-073191264.

119.Hudak P. (1989), “Conception, evolution, and application of functional programming languages”. ACM Computing Surveys Volume 21 Issue 3, pp. 359–411.

120.Cipresso T. (2009). "Software Reverse Engineering Education", Master's Thesis, San Jose State University, http://reversingproject.info/ (Last Accessed 03/03/2011)

121.Object Management Group (2007), “Meta Object Facility (MOF) 2.0 Query/View/Transformation specification”, Revision 1.1, Technical Report, http://www.omg.org/spec/QVT/1.1 (Last checked 05/03/2011)

122.Goldberg K. H.(2009), “XML: Learn XML the Quick and Easy Way”, Second Edition, Peachpit Press, ISBN-13:978-0-321-55967-88, pp. 5-6 explain the rules of well formed XML.

123.Novatchev D. (2001), “The Functional Programming Language XSLT - A proof through examples”, FXSL, the Functional Programming Library for XSLT, Sourceforge portal, http://fxsl.sourceforge.net/articles/FuncProg/Functional%20Programming.html (Last Accessed 05/03/2011)

124.Bell P. (2007), “Application Generation: The limits of XML”, web blog, URL: http://www.pbell.com/index.cfm/2007/1/28/The-Limits-of-XML (Last Accessed 18/05/2011)

125.Jelliffe R. (2007), “Is XML human readable?”, O' Reilly Web blog discussion published on August 8, 2007, URL: http://www.oreillynet.com/xml/blog/2007/08/is_xml_human_readable.html (Last Accessed 18/05/2011)

126.Jones B., Rajabi Z. (2005), “Readability vs. Performance of XML formats”, Discussion on the MSDN web blog., http://blogs.msdn.com/brian_jones/archive/2005/06/23/432018.aspx (Last Accessed 18/05/2011)

127.Bos B. (2003), “What is a good standard? An essay on W3C's design principles”, W3C portal, http://www.w3.org/People/Bos/DesignGuide/introduction, http://www.w3.org/People/Bos/DesignGuide/readability.html (Last Accessed 18/05/2011)

128.Aiken P., Allen D. M. (2004), “XML in Data Management – Understanding and applying them together”, Morgan Kaufmann Publishers, Elsevier, ISBN: 112045599 page 77 of the Chapter “XML Design considerations” discusses the human readability issues.

129.Specification for the Extensible Configuration Checklist Description Format (XCCDF). NISTIR [NIST Interagency Report] #7188. January 2005. Author: Neal Ziring (Information Assurance Directorate, National Security Agency, Fort Meade, MD, USA). Edited by John Wack (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA), http://xml.coverpages.org/NIST-XCCDFv10.pdf (Last Accessed 18/05/2011).

130.Moses, T. (2004), "eXtensible Access Control Markup Language (XACML) Version 2.0. Committee draft 04, 6 December 2004", http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-04.pdf (Last Accessed 18/05/2011)

131.XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999, W3C portal, http://www.w3.org/TR/xslt

132.XML Path Language (Xpath) Version 1.0 W3C Recommendation 16 November 1999, W3C Portal, http://www.w3.org/TR/xpath/

133.XML Schema W3C specifications, W3C Portal, http://www.w3.org/XML/Schema

134.Aho A, Lam M., Sethi R., Ullman J. (2006), "Compilers: Principles, Techniques, and Tools", Pearson International Edition, ISBN 0-321-48681-1.

135.The XML::Twig module at the Perl Comprehensive Archive Network (CPAN), http://search.cpan.org/~mirod/XML-Twig-3.37/Twig.pm

136.Definition of a data model, Wikipedia, URL: http://en.wikipedia.org/wiki/Data_model

137.Definition of domain or "domain of discourse" in Wikipedia, http://en.wikipedia.org/wiki/Domain_of_discourse

138.Berners-Lee T., Fielding R., Masinter R.(2005), “Uniform Resource Identifier (URI): Generic Syntax”, Request for Comments (RFC): 3986, IETF, http://www.ietf.org/rfc/rfc3986.txt

139.Postel J., Reynolds J.(1985), “File Transfer Protocol:FTP”, Request for Comments (RFC):959,IETF, http://www.ietf.org/rfc/rfc959.txt

140.netstat utility web page, http://www.netstat.net/

141.Davison B., Hirsh H. (1998). “Predicting Sequences of User Actions”, Working Notes of the Joint Workshop on Predicting the Future: AI Approaches to Time Series Analysis, Fifteenth National Conference on Artificial Intelligence (AAAI98)/Fifteenth International Conference on Machine Learning (ICML98).

142.Mernik M., Heering J., Sloane A.M. (2005). “When and how to develop domain-specific languages”, ACM Computing Surveys Volume 37, Issue 4, pp. 316–344.

143. Kosar T., L´pez P.E.M., Barrientos P.A., Mernik M. (2008), “A preliminary study on various implementation approaches of domain-specific language.” Information and Software Technology Volume 50, Issue 5, pp. 390–405.

144.White J., Hill J.H., Tambe S., Gokhale A., Schmidt D.C. (2009), “Improving domain- specific language reuse with software product line techniques”, IEEE Software Volume 26, Issue 4 , pp. 47–53.

145. Gabriel P.,Goulao M., Vasco A. (2009), “Do Software Languages Engineers Evaluate their Languages? ”, CITI, Departamento de Inform´tica, Faculdade de Ciencias e Tecnologia, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal.

146. Flegel, U. (2007), “Privacy-Respecting Intrusion Detection”, Advances in Information Security, vol. 35, Springer, New York (2007), ISBN: 978-0-0387-34346-4.

147. Powers S., Peek J., O' Reilly T., Loukides M. (2003), “Unix Power Tools”, O' Reilly Press, 3rd Edition, ISBN: 978-0596-00330-2.

148. Lite J. (2009), “Porn among National Science Foundation's 'research' “, Scientific American, news blog entry, http://www.scientificamerican.com/blog/post.cfm?id=porn-among-national-science-foundat-2009-01-29 (Last accessed 18/03/2011)

149. Wikipedia (2011), “Forward-confirmed reverse DNS” page, http://en.wikipedia.org/wiki/Forward-confirmed_reverse_DNS (Last accessed 18/03/2011)

150. The Mozillazine (2009), “Places.sqlite”, article describing the contents of the Mozilla Firefox web history recording mechanism, http://kb.mozillazine.org/Places.sqlite (Last accessed 18/03/2011)

151. Shepler S., Callaghan B., Robinson D., Thurlow R, Beame C., Eisler M., Noveck D. (2003), “Network File System (NFS) version 4 Protocol”, IETF Request For Comments RFC 3530.

152.Hertel C. (2003) “Implementing CIFS — The Common Internet FileSystem”, Prentice Hall. ISBN 0-13-047116-X.

153.Willard N. (2007), “Malware and Porn Traps: What Schools Can Do”, article on the Education World web portal, date 27/02/2007, http://www.educationworld.com/a_tech/columnists/willard/willard005.shtml (Last accessed 19/03/2011)

154.Posey B. (2006), “The importance of an effective software installation policy”, TechRepublic web portal, http://www.techrepublic.com/article/the-importance-of-an-effective-software-installation-policy/6068213 (Last accessed 27/03/2011)

155.Microsoft TechNet (2004), “Using Software Restriction Policies to Protect Against Unauthorized Software”, http://technet.microsoft.com/en-us/library/bb457006.aspx (Last accessed 27/03/2011)

156.Handley M., Rescorla E. (2006), “Internet Denial of Service Considerations”, IETF Request for Comment (RFC) 4732.

157.Addley E., Halliday J. (2010), “Operation Payback cripples MasterCard site in revenge for WikiLeaks ban”, The Guardian newspaper web portal, http://www.guardian.co.uk/media/2010/dec/08/operation-payback-mastercard-website-wikileaks (Last accessed 28/03/2011)

158.LOIQ DDoS tool for Linux development at Sourceforge, http://sourceforge.net/projects/loiq/(Last Accessed 28/03/2011)

159.Abbadi I., Alawneh M. (2008), “Preventing Insider Information Leakage for Enterprises”, The Second International Conference for Emerging Security Information, Systems and Technologies, IEEE Computer Society, DOI 10.1109/SECURWARE.2008.14 .

160.Onno S. (2008), "A Federated Physical and Logical Access Control Enforcement Model," ares, Third International Conference on Availability, Reliability and Security, pp. 683-692.

161. Loscocco P., Smalley S., Muckelbauer P., Taylor R., Turner S., Farrell J. (1998), “The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments”, In Proceedings of the 21st National Information Systems Security Conference, pp. 303–314.

162.National Security Agency (NSA) (2009), “Security-Enhanced Linux”, http://www.nsa.gov/research/selinux/index.shtml (Last Accessed 31/03/2011)

163.Rilley S. (2006), “Mandatory integrity control in Windows Vista”, Microsoft Technet Blogs, http://blogs.technet.com/b/steriley/archive/2006/07/21/442870.aspx (Last accessed 31/03/2011)

164.The 'snoopy' execve logger and wrapper at Sourceforge, http://sourceforge.net/projects/snoopylogger/ (Last Accessed 02/04/2011)

165. Ollmann G. (2006), “The Phising Guide: Understanding and Preventing Phising Attacks”, http://www.technicalinfo.net/papers/Phishing.html (Last Accessed 15/05/2011)

166. Song D. (2000), “dsniff”, dsniff is a collection of tools for network auditing and penetration testing, http://monkey.org/~dugsong/dsniff/ (Last Accessed 15/05/2011)

167. Phyo A., Furnell S. (2003), “Data Gathering for Insider Misuse Monitoring”, Proceedings of the 2nd European Conference on Information Warfare and Security, Reading, UK, 30 June - 1 July, pp247-254.

168. The Transaction Processing Performance Council website (2011), http://www.tpc.org, (Last Accessed 12/11/2011)

169. Strandell T. (2010), “Open Source Database Systems: Systems Study, Performance and Scalability”, VDM Verlag Dr. Müller, ISBN: 978-3639093506.

170. Zawodny J. (2009), “NoSQL: Distributed and Scalable Non-Relational Database Systems”, Linux Magazine web portal, http://www.linux-mag.com/id/7579/ (Last Accessed 12/11/2011)

171. Vogels W. (2008), “Eventually Consistent”, ACM Queue Magazine, Volume 6 Issue 6, October 2008, http://dx.doi.org/10.1145/1466443.1466448 (Last Accessed 12/11/2011).

172.Westin A. (1987), “Privacy and Freedom”, Bodley Head, New York.

173.European Parliament (1995), Directive 95/46/EC: On the protection of individuals with regard to the processing of personal data and on the free movement of such data, Official Journal L 281, October 1995, http://europa.eu.int/eur-lex/en/lif/dat/1995/en_395L0046.html (Last Accessed 18/03/2011)

174.Biskup J., Karabulut Y. (2002), “A hybrid PKI model with an application for secure mediation”, Proceedings of the 16th Annual IFIP WG 1.3 Working Conference on Data and Application Security, Cambridge, England, July 2002, pp. 271-282.

175.Biskup J., Karabulut Y. (2003), “Mediating between strangers: A trust management based approach.”, Proceedings of the 2nd Annual PKI Research Workshop, Gaitherburg, Maryland, USA, April 2003, pp. 80-95.

176.Karabulut Y. (2002), “Secure Mediation Between Strangers in Cyberspace”, PhD Thesis, University of Dortmund, Germany.

177.Schultz G. (2009), “The Green and Virtual Data Center”, CRC Press, Taylor & Francis Group, ISBN: 978-1-4200-8666-9.

178.Sailer R., Valdez E., Jaeger T., Perez R., Van Doorn L., Griffin J., Berger S. (2005), “sHype: Secure Hypervisor Approach to Trusted Operating Systems”, IBM Research Report RC23511 (W0502-006), February 2, 2005, Thomas J. Watson Research Center.

179. Azab A., Ning P., Zhang X. (2011), “SICE: a hardware-level strongly isolated computing environment for x86 multi-core platforms”, Proceedings of the 18th ACM conference on Computer and communications security, ISBN: 978-1-4503-0948-6, pp.375-388.

Appendix A : LUARM sample source code

This appendix contains the PERL source code of the LUARM client monitoring modules for reference. The complete source code of the LUARM/ITPSL system can be found on-line at:

http://luarm.svn.sourceforge.net/viewvc/luarm/

and also in the accompanying thesis DVD. The reader should prefer the online reference, in order to obtain up-to-date code versions of the system.

```
#!/usr/bin/perl

#fileactivity.pl - By George Magklaras - Center for Security, Communications and Networks Research - University of
Plymouth, UK
#Synopsis: This program populates the ITPSL RDBMS with file access data.

use Time::HiRes qw(usleep clock_gettime gettimeofday clock_getres CLOCK_REALTIME ITIMER_REAL
ITIMER_VIRTUAL ITIMER_PROF ITIMER_REALPROF);
use Digest::MD5 qw(md5 md5_hex md5_base64);
use DBI;
use strict;

my $VERSION="alpha1";

#Some essential sanity checks
my @whoami=getpwuid($<);
die "fileactivity.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);


#System variables
#What's the sampling frequency in microseconds
my $sdelay=100000;

#Get the pid of this fileactivity.pl instance (startluarmclient.pl and stopluarmclient.pl scripts need this
my $fapid="$$";
system "echo $fapid > managescripts/.fapid";


#And now we fo into a sampling loop
while (1==1) {
		usleep($sdelay);
		my $loopout=`lsof -w -n -M -P | grep -v IPv4 | grep -v IPv6 | grep -v ^COMMAND | grep -v lsof | grep -v
"can't" | grep -v "socket" | grep -v "/lib64/" | grep -v "/lib/" | grep -v "FIFO" | grep -v "root" | grep "REG" | grep -v
"mem" | grep -v "rpcbind" | grep -v "dbus-daemon"`;
		my @looparray=split("\n",$loopout);
		dbfileupdate(\@looparray);
		chnonexisting();
}
```

```
#Subroutine definitions here

#Essential for the way ITPSL specifies the file (filename and location)
#Takes as an argument a refence to a string and returns a list of two strings
#that represent the location (directory) and the filename.
sub splitnameloc {
        my $reftostring=shift;
        my @locname=split("/", $$reftostring);
        #The filename has to be the last array element)
        my $filename=pop @locname;
        #Then what's
        my $location=join("/",@locname);
        return ($filename,$location);
}

sub timestamp {
        #get the db authentication info
        my @authinfo=getdbauth();
        my ($username,$dbname,$dbpass,$hostname);

        foreach my $dbentry (@authinfo) {
              ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);                  }
        #Connect to the LUARM RDBMS server and get the timestamp from MySQL
        my $datasource="DBI:mysql:$dbname:$hostname:3306";
        my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});
        my $SQLh=$itpslservh->prepare("select DATE_FORMAT(NOW(), '%Y-%m-%d-%k-%i-%s')");
        $SQLh->execute();
        my @timearray=$SQLh->fetchrow_array();
        my ($year,$month,$day,$hour,$min,$sec)=split("-",$timearray[0]);
        $SQLh->finish();
        return ($year,$month,$day,$hour,$min,$sec);
}


sub dbfileupdate  {

        my $reftoarray=shift;

        #get the db authentication info
        my @authinfo=getdbauth();
        my ($username,$dbname,$dbpass,$hostname);

        foreach my $dbentry (@authinfo) {
                ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
        }

        my $datasource="DBI:mysql:$dbname:$hostname:3306";
        my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

        #Calculate the MD5 hex checksum of each record
        foreach my $record (@$reftoarray) {
                my $md5s=md5_hex($record);
                #Select only the records that are relevant (i.e. not closed)
                my $SQLh=$itpslservh->prepare("SELECT md5sum FROM fileinfo WHERE
md5sum='$md5s' AND dyear is NULL and dmonth is NULL");
                $SQLh->execute();
                my @md5hits=$SQLh->fetchrow_array();
                if ($md5hits[0] eq $md5s) { $SQLh->finish();
```

```
    } elsif (!defined($md5hits[0])) {
        my($command,$pid,$user,$fd,$type,$devperlprogsice,$size,$node,$name)=split(" ", $record);
        my @nameloc;
        #This if statement addresses sourceforge bug 3024339. Because the lsof output can
        #omit the size-off output for certain types of files, we could have an empty $name
        #This could break the SQL INSERT statement (non NULL column). For that reason, we
perform
        #the following changes. If size-off is empty and $name is not defined due to the way we parse
        #we exchange $node and name vars and we solve the problem.
        if ($name) {
            @nameloc=splitnameloc(\$name); } else {
            $name=$node;
            $node=0;
            @nameloc=splitnameloc(\$name);
        }
        my ($cyear,$cmonth,$cday,$chour,$cmin,$csec)=timestamp();
        #Quote fields that might contain characters that need escaping, to address sourceforge bug
3024339.
        $nameloc[0]=$itpslservh->quote($nameloc[0]);
        $nameloc[1]=$itpslservh->quote($nameloc[1]);

        my $rows=$itpslservh->do ("INSERT INTO fileinfo
(md5sum,filename,location,username,application,fd,pid,size,cyear,cmonth,cday,chour,cmin,csec)"
                  . "VALUES ('$md5s',$nameloc[1],
$nameloc[0],'$user','$command','$fd','$pid','$size',"
                  . "'$cyear','$cmonth','$cday','$chour','$cmin','$csec')" );

        if (($rows==-1) || (!defined($rows))) {
            print "fileactivity.pl Fatal Error: dbfileupdate : No records were altered. Record
$record was not registered.\n";
        }
        #Finally close the db handler
        $SQLh->finish();
}

    }


}
sub chnonexisting {

    #get the db authentication info
    my @authinfo=getdbauth();
    my ($username,$dbname,$dbpass,$hostname);

    foreach my $dbentry (@authinfo) {
        ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
    }

    my $datasource="DBI:mysql:$dbname:$hostname:3306";
    my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});


    #Get all the MD5sums of the records in the database that are not closed
    my $SQLh=$itpslservh->prepare("SELECT md5sum FROM fileinfo WHERE dyear is NULL AND
dmonth is NULL");
    $SQLh->execute();
    my @md5hits;
```

```
		while (my $row=$SQLh->fetchrow_array()) {
				push(@md5hits, $row);
		#With this method we retrieve multiple results
		}

		#Now sample the current file activity and generate the live md5sums for each file record
		my $loopout=`lsof -w -n -M -P | grep -v IPv4 | grep -v IPv6 | grep -v ^COMMAND | grep -v lsof | grep -v
"can't" | grep -v "socket" | grep -v "/lib64/" | grep -v "/lib/"| grep -v "FIFO" | grep -v "root" | grep "REG" | grep -v
"mem" | grep -v "rpcbind" | grep -v "dbus-daemon"`;
		my @looparray=split("\n",$loopout);
		my @livemd5;
		foreach my $liverecord (@looparray) {
				my $md5live=md5_hex($liverecord);
				push(@livemd5, $md5live);
		}

		#For each md5 record obtained from RDBMS, check if it exists in the live..
		#If it doesn't timestamp the closing time table columns for the record in the RDBMS.
		#To do that, first take the difference between the two arrays (the difference of the md5hits to the livemd5)
		my @arrunion=grep!${{map{$_,1}@livemd5}}{$_},@md5hits;
		foreach my $tobeclosed (@arrunion) {
				my ($dyear,$dmonth,$dday,$dhour,$dmin,$dsec)=timestamp();
				my $rows=$itpslservh->do ("UPDATE fileinfo set
dyear='$dyear',dmonth='$dmonth',dday='$dday',dhour='$dhour',dmin='$dmin',dsec='$dsec'"
										. " where md5sum='$tobeclosed'" );
		}#end of foreach $tobeclosed
		$SQLh->finish();

}#end of chknoexisting

sub getdbauth {
		unless(open DBAUTH, "</root/itpsl/.adb.dat") {
						die "endpointresolver.pl Error:getdbauth: Could not open the .adb.dat file due to:
$!";
				}

		my @localarray;

		while (<DBAUTH>) {
				my $dbentry=$_;
				chomp($dbentry);
				push(@localarray, $dbentry);
		}

		return @localarray;

} #end of getdbauth()
```

```
#!/usr/bin/perl
```

```
#netactivity.pl - By George Magklaras - Center for Security, Communications and Networks Research - University of
Plymouth, UK
#Synopsis: This program populates the ITPSL RDBMS with network endpoint data.

use Time::HiRes qw(usleep clock_gettime gettimeofday clock_getres CLOCK_REALTIME ITIMER_REAL
ITIMER_VIRTUAL ITIMER_PROF ITIMER_REALPROF);
use Digest::MD5 qw(md5 md5_hex md5_base64);
use DBI;
use strict;

my $VERSION="alpha1";

#Some essential sanity checks
my @whoami=getpwuid($<);
die "netactivity.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);


#The database authentication file contains the authentication info required to connect
#to each host database (one database per monitored host): username, database_name, database_password
die "endpointresolver.pl Error:No database authentication file found. \n"
if (! (-e "/root/itpsl/.adb.dat"));

#Get the pid of this netactivity.pl instance (startluarmclient.pl and stopluarmclient.pl scripts need this
my $napid="$$";
system "echo $napid > managescripts/.napid";

#System variables
#What's the sampling frequency in microseconds
my $sdelay=100000;

my $filterstring="lsof -i -n -w -P | grep -v ^COMMAND";

#And now we fo into a sampling loop
while (1==1) {
		usleep($sdelay);
		my $loopout=`$filterstring`;
		my @looparray=split("\n",$loopout);
		dbfileupdate(\@looparray);
		chnonexisting();
}
#Subroutine definitions here

sub timestamp {
		#get the db authentication info
		my @authinfo=getdbauth();
		my ($username,$dbname,$dbpass,$hostname);

		foreach my $dbentry (@authinfo) {
			($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);				}

		my $datasource="DBI:mysql:$dbname:$hostname:3306";
		#Connect to the LUARM RDBMS server and get the timestamp from MySQL
		my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});
		my $SQLh=$itpslservh->prepare("select DATE_FORMAT(NOW(), '%Y-%m-%d-%k-%i-%s')");
		$SQLh->execute();
		my @timearray=$SQLh->fetchrow_array();
		my ($year,$month,$day,$hour,$min,$sec)=split("-",$timearray[0]);
```

```
        $SQLh->finish();
        return ($year,$month,$day,$hour,$min,$sec);

}


sub splitipdata {
        my $refipdata=shift;
        #Now we have to see what type of connection info we have (LISTEN, ESTABLISHED, IPv4/6)
        #as all these have a different type of structure
        if ($$refipdata =~/.+->.+/) {
                #We have an established TCPv4 connection
                my ($source,$destination)=split("->",$$refipdata);
                my ($sourceip,$sourceport)=split(":",$source);
                my ($destip,$destport)=split(":",$destination);
                return ($sourceip,$sourceport,$destip,$destport);
                } elsif ($$refipdata =~/.*:d+/) {
                #This must be a UDP or TCP LISTEN endpoint
                my ($sourceip, $sourceport)=split(":",$$refipdata);
                if ($sourceip eq "*") {
                        $sourceip="ALL";
                }
                my $destip="ALL";
                my $destport="ALL";
                return ($sourceip,$sourceport,$destip,$destport);
                }
}

sub dbfileupdate  {

        my $reftoarray=shift;

        #get the db authentication info
        my @authinfo=getdbauth();
        my ($username,$dbname,$dbpass,$hostname);

        foreach my $dbentry (@authinfo) {
                ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
        }

        my $datasource="DBI:mysql:$dbname:$hostname:3306";
        my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});
        #my @arrayofchecksums;
        #Calculate the MD5 hex checksum of each record
        foreach my $record (@$reftoarray) {

                my ($application,$pid,$username,$fd,$ipversion,$device,$size,$transport,$name)=split(" ",
$record);

                #now split the sourceip,destip,sourceport,destport from $name into separate variables
                my ($sourceip,$sourceport,$destip,$destport)=splitipdata(\$name);
                my $md5s=md5_hex(join("," , $sourceip,$sourceport,$destip,$destport,$application,$pid,
$username,$transport));
                #Does it exist and is it relevant? This query matches all fields and should return 1 only result if
we get things right
                my $SQLh=$itpslservh->prepare("SELECT COUNT(*) FROM netinfo WHERE
md5sum='$md5s' ");
                $SQLh->execute();
```

```
				my @md5hits=$SQLh->fetchrow_array();

				if ( $md5hits[0]=="1") { #The record exists
								$SQLh->finish();
		} elsif ($md5hits[0]=="0") {
				#New record, we have to insert it.

				#In order to avoid passing empty values for the source
				#IP and dest IP that would terminate prematurely a
				#fetchrow_array OP in endpointresolver.pl we have to
				#put NONE to both fields here.
				if ($sourceip eq "") {
						$sourceip="NONE";
				}

				if ($destip eq "") {
						$destip="NONE";
				}

				my ($cyear,$cmonth,$cday,$chour,$cmin,$csec)=timestamp();
				my $rows=$itpslservh->do ("INSERT INTO netinfo (md5sum,
sourceip,sourceport,destip,destport,application,ipversion,pid,username,transport,cyear,cmonth,cday,chour,cmin,csec)"
									. "VALUES
('$md5s','$sourceip','$sourceport','$destip','$destport','$application','$ipversion','$pid','$username','$transport',"
									. "'$cyear','$cmonth','$cday','$chour','$cmin','$csec')" );

				if (($rows==-1) || (!defined($rows))) {
						print "netactivity.pl Fatal Error: dbfileupdate : No records were altered. Record
$record was not registered.\n";
				}
				#Finally close the db handler
				$SQLh->finish();
}

		}


}

sub chnonexisting {
		#get the db authentication info
		my @authinfo=getdbauth();
		my ($username,$dbname,$dbpass,$hostname);

		foreach my $dbentry (@authinfo) {
				($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
		}

		my $datasource="DBI:mysql:$dbname:$hostname:3306";
		my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});
		#Get all the network endpoint records in the database that are not closed
		my $SQLh=$itpslservh->prepare("SELECT md5sum FROM netinfo WHERE dyear is NULL AND dmonth
is NULL AND dday is NULL");
		$SQLh->execute();
		my @md5hits;
		while (my $row=$SQLh->fetchrow_array()) {
				push(@md5hits, $row);
		#With this method we retrieve multiple results
		}

		#Now sample the current endpoint activity.
		my $loopout=`$filterstring`;
		my @looparray=split("\n",$loopout);
		my @livemd5;

		foreach my $liverecord (@looparray) {

				my ($application,$pid,$username,$fd,$ipversion,$device,$size,$transport,$name)=split(" ",
$liverecord);

				#now split the sourceip,destip,sourceport,destport from $name into separate variables
				my ($sourceip,$sourceport,$destip,$destport)=splitipdata(\$name);
				my $md5s=md5_hex(join("," , $sourceip,$sourceport,$destip,$destport,$application,$pid,
$username,$transport));
				push(@livemd5, $md5s);
		}

		#For each md5 record obtained from RDBMS, check if it exists in the live..
		#If it doesn't timestamp the closing time table columns for the record in the RDBMS.
		#To do that, first take the difference between the two arrays (the difference of the md5hits to the livemd5)
		my @arrunion=grep!${{map{$_,1}@livemd5}}{$_},@md5hits;

		foreach my $tobeclosed (@arrunion) {
				my ($dyear,$dmonth,$dday,$dhour,$dmin,$dsec)=timestamp();
				my $rows=$itpslservh->do ("UPDATE netinfo set
dyear='$dyear',dmonth='$dmonth',dday='$dday',dhour='$dhour',dmin='$dmin',dsec='$dsec'"
										. " where md5sum='$tobeclosed'" );
		}#end of foreach $tobeclosed
		$SQLh->finish();
}#end of chknoexisting

sub getdbauth {
		unless(open DBAUTH, "</root/itpsl/.adb.dat") {
						die "endpointresolver.pl Error:getdbauth: Could not open the .adb.dat file due to:
$!";
				}

		my @localarray;

		while (<DBAUTH>) {
				my $dbentry=$_;
				chomp($dbentry);
				push(@localarray, $dbentry);
		}

		return @localarray;

} #end of getdbauth()
```

```
#!/usr/bin/perl

#psactivity.pl - By George Magklaras - Center for Security, Communications and Networks Research - University of
Plymouth, UK
#Synopsis: This program populates the ITPSL RDBMS with process execution data and it effectively makes the need of
deploying
#execve wrappers obsolete.

use Time::HiRes qw(usleep clock_gettime gettimeofday clock_getres CLOCK_REALTIME ITIMER_REAL
ITIMER_VIRTUAL ITIMER_PROF ITIMER_REALPROF);
use Digest::MD5 qw(md5 md5_hex md5_base64);
use DBI;
use strict;
use Fcntl qw(:DEFAULT :flock);

my $VERSION="alpha1";

#Some essential sanity checks
my @whoami=getpwuid($<);
die "psactivity.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

#Get the pid of this psactivity.pl instance (startluarmclient.pl and stopluarmclient.pl scripts need this
my $pspid="$$";
system "echo $pspid > managescripts/.pspid";



#System variables
#What's the sampling frequency in microseconds
my $sdelay=10;

#And now we fo into a sampling loop
while (1==1) {
        sysopen(FH, "/tmp/.psdata", O_RDWR ) or die "psactivity.pl Error:can't open the psdata file: $! \n";
        #Lock the file so that it does not get corrupted.
        #flock(FH, LOCK_EX) or die "psactivity.pl Error:can't lock the psdata file: $!\n";
        my $loopstring = <FH>;

        truncate(FH, 0) or die "psactivity.pl Error: can't empty psdata: $! \n";
        close(FH);
        my @looparray=split("\n",$loopstring);
        dbprocupdate(\@looparray);
        chnonexisting();
        usleep($sdelay);
}
#Subroutine definitions here

#Essential for the way ITPSL specifies the process (command and argument(s))
#Takes as an argument a refence to a string and returns a list of two strings
#that represent the process name and its respective arguments.
sub splitcommandargs {
        my $reftostring=shift;
        my @proc=split(" ", $$reftostring);
        #The command name has to be the first array element
        my $commname=shift @proc;
        #Whatever is left are the arguments to the command.
        my $arguments=join(" ",@proc);
        return ($commname,$arguments);
}

sub timestamp {
        #get the db authentication info
        my @authinfo=getdbauth();
        my ($username,$dbname,$dbpass,$hostname);

        foreach my $dbentry (@authinfo) {
                ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
        }

        my $datasource="DBI:mysql:$dbname:$hostname:3306";
        my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

        #Get all the MD5sums of the records in the database that are not closed
        my $SQLh=$itpslservh->prepare("select DATE_FORMAT(NOW(), '%Y-%m-%d-%k-%i-%s')");
        $SQLh->execute();

        my @timearray=$SQLh->fetchrow_array();
        my ($year,$month,$day,$hour,$min,$sec)=split("-",$timearray[0]);
        $SQLh->finish();
        return ($year,$month,$day,$hour,$min,$sec);
}

sub dbprocupdate  {

        my $reftoarray=shift;

        #get the db authentication info
        my @authinfo=getdbauth();
        my ($username,$dbname,$dbpass,$hostname);

        foreach my $dbentry (@authinfo) {
                ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
        }

        my $datasource="DBI:mysql:$dbname:$hostname:3306";
        my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

        #Calculate the MD5 hex checksum of each record
        foreach my $record (@$reftoarray) {
                my ($pid,$ppid,$uid,$command,$comwargs)=split(",",$record);
                #Now split the process name (command) from its arguments
                my ($comm,$arguments)=splitcommandargs(\$comwargs);
                #And then produce the md5sum of the record ONLY FROM SOME FIELDS THAT REMAIN
STATIC
                #In this case, the md5sum will be the pid, the ppid, the username, and the command
                my $username=getpwuid($uid);
                my $md5s=md5_hex(join("," , $username,$pid,$ppid,$command));
                #Does this process exist in the DB and is it relevant? This should return 1 only result if we get
things right.
                my $SQLh=$itpslservh->prepare("SELECT COUNT(*) FROM psinfo WHERE
md5sum='$md5s' AND dyear is NULL AND dmonth is NULL ");
                $SQLh->execute();
                my @md5hits=$SQLh->fetchrow_array();
                #Select only the records that are relevant (i.e. not closed)
                #Does the record exist?
```

```
				if ($md5hits[0]=="1") {
						#Record exists.

						#$SQLh->finish();
			} elsif ( $md5hits[0]=="0") {
				#The record does not exist. We need to SQL INSERT it.
				my ($cyear,$cmonth,$cday,$chour,$cmin,$csec)=timestamp();
				#Does the @arguments string contain a single quote character?
				#sourceforge Bug 3024339. If yes, we need to escape this properly
				#by using quote.
				$arguments=$itpslservh->quote($arguments);
				my $rows=$itpslservh->do ("INSERT INTO psinfo
(md5sum,username,pid,ppid,command,arguments,cyear,cmonth,cday,chour,cmin,csec)"
										. "VALUES ('$md5s','$username','$pid','$ppid','$command',
$arguments,"
										. "'$cyear','$cmonth','$cday','$chour','$cmin','$csec')" );

				if (($rows==-1) || (!defined($rows))) {
						print "fileactivity.pl Fatal Error: dbfileupdate : No records were altered. Record
$record was not registered.\n";
				}
				#Finally close the db handler
				$SQLh->finish();
}

			}


}

sub chnonexisting {

		sysopen(FH, "/tmp/.psdata", O_RDONLY ) or die "psactivity.pl Error:can't open the psdata file: $! \n";
		#Lock the file so that it does not get corrupted.
		flock(FH, LOCK_EX) or die "psactivity.pl Error:can't lock the psdata file: $!\n";
		my $loopstring = <FH>;
		close(FH);
		my @looparray=split("\n",$loopstring);

		#get the db authentication info
		my @authinfo=getdbauth();
		my ($username,$dbname,$dbpass,$hostname);

		foreach my $dbentry (@authinfo) {
				($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
		}

		my $datasource="DBI:mysql:$dbname:$hostname:3306";
		my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

		#Get all the MD5sums of the records in the database that are not closed
		my $SQLh=$itpslservh->prepare("SELECT md5sum FROM psinfo WHERE dyear is NULL AND dmonth
is NULL");
		$SQLh->execute();
		my @md5hits;
		while (my $row=$SQLh->fetchrow_array()) {
				push(@md5hits, $row);
		#With this method we retrieve multiple results
```

```
    }

    #Now generate the live md5sums for each file record
    my @livemd5;
    foreach my $record (@looparray) {
        my ($pid,$ppid,$uid,$command,$comwargs)=split(",",$record);
        #Now split the process name (command) from its arguments
        my ($comm,$arguments)=splitcommandargs(\$comwargs);
        #And then produce the md5sum of the record ONLY FROM SOME FIELDS THAT REMAIN
STATIC
        #In this case, the md5sum will be the pid, the ppid, the username, and the command
        my $username=getpwuid($uid);
        push(@livemd5, md5_hex(join("," , $username,$pid,$ppid,$command)));

    }

    #For each md5 record obtained from RDBMS, check if it exists in the live..
    #If it doesn't timestamp the closing time table columns for the record in the RDBMS.
    #To do that, first take the difference between the two arrays (the difference of the md5hits to the livemd5)
    my @arrunion=grep!${{map{$_,1}@livemd5}}{$_},@md5hits;

    foreach my $tobeclosed (@arrunion) {
        my ($dyear,$dmonth,$dday,$dhour,$dmin,$dsec)=timestamp();
        my $rows=$itpslservh->do ("UPDATE psinfo set
dyear='$dyear',dmonth='$dmonth',dday='$dday',dhour='$dhour',dmin='$dmin',dsec='$dsec'"
                      . " where md5sum='$tobeclosed'" );
    }#end of foreach $tobeclosed

    $SQLh->finish();
}#end of chknoexisting

sub getdbauth {
    unless(open DBAUTH, "</root/itpsl/.adb.dat") {
            die "endpointresolver.pl Error:getdbauth: Could not open the .adb.dat file due to:
$!";
        }

    my @localarray;

    while (<DBAUTH>) {
        my $dbentry=$_;
        chomp($dbentry);
        push(@localarray, $dbentry);
    }

    return @localarray;

} #end of getdbauth()
```

```
#!/usr/bin/perl

#hwactivity.pl - By George Magklaras - Center for Security, Communications and Networks Research - University of
Plymouth, UK
```

```
#Synopsis: This program populates the ITPSL RDBMS with hardware device connection data. All hardware PCI and
USB devices present
#to the system are logged in terms of their insertion and removal time.

use Time::HiRes qw(usleep clock_gettime gettimeofday clock_getres CLOCK_REALTIME ITIMER_REAL
ITIMER_VIRTUAL ITIMER_PROF ITIMER_REALPROF);
use Digest::MD5 qw(md5 md5_hex md5_base64);
use DBI;
use strict;

my $VERSION="alpha1";

#Some essential sanity checks
my @whoami=getpwuid($<);
die "hwactivity.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);


#System variables
#What's the sampling frequency in microseconds
my $sdelay=4000000;

#Get the pid of this hwactivity.pl instance (startluarmclient.pl and stopluarmclient.pl scripts need this
my $hwpid="$$";
system "echo $hwpid > managescripts/.hwpid";


#We start with the PCI devices
#and use the -vm flag for backwards compatibility
#see lspci manual page.
#The perl filter removes the lines that *BEGIN* with \n to parse the data properly.
my $hwfilterstring1="lspci -vm | grep -2 Class | perl -pi -w -e 's/^\n//g;' ";

#We also have to check the USB bus
my $hwfilterstring2="lsusb -v | grep -1 idVendor";

#And who was logged in at that time for accountability
my $userslogged="users";

#And now we go into a sampling loop
while (1==1) {
                usleep($sdelay);
                my $pcidata=`$hwfilterstring1`;
                my $usbdata=`$hwfilterstring2`;
                my $loggedusersdata=`$userslogged`;
                my @pciarray=split("--\n",$pcidata);
                my @usbarray=split("--\n",$usbdata);
                my @userarray=split(" ", $loggedusersdata);
                #Make sure that the userarray element has no duplicate user entries
                #(users might be logged in in multiple pts, ttys)
                my %seenuser=();
                my @uniquserarray=();
                foreach my $user (@userarray) {
                    unless ($seenuser{$user}) {
                            $seenuser{$user}=1;
                            push(@uniquserarray, $user);
                            }
                } #end of foreach
```

```
		#Debug
		#print "pciarray0 is $pciarray[0], pciarray[1] is $pciarray[1], usbarray[0] is $usbarray[0] and usbarray[1] is
$usbarray[1] \n";
		dbprocupdatehw(\@pciarray,\@usbarray,\@uniquserarray);
		chnonexistinghw();
}
#Subroutine definitions here

sub timestamp {
		#get the db authentication info
		my @authinfo=getdbauth();
		my ($username,$dbname,$dbpass,$hostname);

		foreach my $dbentry (@authinfo) {
			($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
					}
		#Connect to the LUARM RDBMS server and get the timestamp from MySQL
		my $datasource="DBI:mysql:$dbname:$hostname:3306";
		my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});
		my $SQLh=$itpslservh->prepare("select DATE_FORMAT(NOW(), '%Y-%m-%d-%k-%i-%s')");
		$SQLh->execute();
		my @timearray=$SQLh->fetchrow_array();
		my ($year,$month,$day,$hour,$min,$sec)=split("-",$timearray[0]);
		$SQLh->finish();
		return ($year,$month,$day,$hour,$min,$sec);

}

#Takes two array references as arguments
sub dbprocupdatehw {
		my $reftopciarray=shift;
		my $reftousbdarray=shift;
		my $reftouserarray=shift;

		#get the db authentication info
		my @authinfo=getdbauth();
		my ($username,$dbname,$dbpass,$hostname);

		foreach my $dbentry (@authinfo) {
				($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
		}

		my $datasource="DBI:mysql:$dbname:$hostname:3306";
		my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

		#Dealing with the PCI bus first
		foreach my $pcirec (@$reftopciarray) {
				my ($dev1,$string1,$vendor,$string2)=split("\n",$pcirec);
				my @devvendor=split(":", $vendor);
				my @devstring1=split(":", $string1);
				my @devstring2=split(":", $string2);
				my $devstring=$devstring1[1].$devstring2[1];
				my $loggedusers=join(",", @$reftouserarray);
				my $devbus="PCI";
				#Calculate the md5sum for the PCI device record
				my $md5s=md5_hex(join("," , $devvendor[1],$devstring,$devbus));
				#Do we have already the device on the hwinfo table of the host?
```

```
        my $SQLh=$itpslservh->prepare("SELECT COUNT(*) FROM hwinfo WHERE
md5sum='$md5s' AND dyear is NULL AND dmonth is NULL ");
        $SQLh->execute();
        my @md5hits=$SQLh->fetchrow_array();
        #Select only the records that are relevant (i.e. not closed)
        #Does the record exist?
        if ($md5hits[0]=="1") {
            #Record exists. Close the DB handler.
            $SQLh->finish();
    } elsif ( $md5hits[0]=="0") {
        #The record does not exist. We need to SQL INSERT it.
        my ($cyear,$cmonth,$cday,$chour,$cmin,$csec)=timestamp();
        #Quote the devicestring as it is likely to contain characters that can break the SQL INSERT
statement
        $devstring=$itpslservh->quote($devstring);
        my $rows=$itpslservh->do ("INSERT INTO hwinfo
(md5sum,cyear,cmonth,cday,chour,cmin,csec,devbus,devstring,devvendor,userslogged)"
                  . "VALUES
('$md5s','$cyear','$cmonth','$cday','$chour','$cmin','$csec','$devbus',$devstring,'$devvendor[1]','$loggedusers')" );
        if (($rows==-1) || (!defined($rows))) {
            print "hwactivity.pl Fatal Error: dbfileupdate : No records were altered. Record
$pcirec was not registered.\n";
        }
        #Finally close the db handler
        $SQLh->finish();

            }#end of elsif

        } #end of foreachloop
    #Now we have to deal with the USB device data
    foreach my $usbrec (@$reftousbdarray) {
        my ($garbage,$vendor,$string)=split("\n",$usbrec);
        my ($v1,$v2,$v3,$v4,$v5,$v6,$v7,$v8)=split(" ",$vendor);
        my $devvendor="$v3 "."$v4 "."$v5 "."$v6 "."$v7 "."$v8";
        my ($s1,$s2,$s3,$s4,$s5,$s6,$s7,$s8)=split(" ",$string);
        my $devstring="$s3 "."$s4 "."$s5 "."$s6 "."$s7 "."$s8";
        my $loggedusers=join(",", @$reftouserarray);
        my $devbus="USB";
        #Calculate the md5sum for the USB device record
        my $md5s=md5_hex(join("," ,$devvendor,$devstring,$devbus));

        #Do we have already the device on the hwinfo table of the host?
        my $SQLh=$itpslservh->prepare("SELECT COUNT(*) FROM hwinfo WHERE
md5sum='$md5s' AND dyear is NULL AND dmonth is NULL ");
        $SQLh->execute();
        my @md5hits=$SQLh->fetchrow_array();

        if ($md5hits[0]=="1") {
            #Record exists. Close the DB handler.
            $SQLh->finish();
    } elsif ( $md5hits[0]=="0") {
        #The record does not exist. We need to SQL INSERT it.
        my ($cyear,$cmonth,$cday,$chour,$cmin,$csec)=timestamp();
        #Quote the devicestring as it is likely to contain characters that can break the SQL INSERT
statement
        $devstring=$itpslservh->quote($devstring);
        my $rows=$itpslservh->do ("INSERT INTO hwinfo
(md5sum,cyear,cmonth,cday,chour,cmin,csec,devbus,devstring,devvendor,userslogged)"
```

```
                  . "VALUES
('$md5s','$cyear','$cmonth','$cday','$chour','$cmin','$csec','$devbus',$devstring,'$devvendor','$loggedusers')" );
        if (($rows==-1) || (!defined($rows))) {
            print "hwactivity.pl Fatal Error: dbfileupdate : No records were altered. Record
$usbrec was not registered.\n";
        }
        #Finally close the db handler
        $SQLh->finish();

            }#end of elsif

        } #end of foreachloop
}#end of dbprocupdatehw

sub chnonexistinghw {
    #get the db authentication info
    my @authinfo=getdbauth();
    my ($username,$dbname,$dbpass,$hostname);

    foreach my $dbentry (@authinfo) {
        ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
    }

    my $datasource="DBI:mysql:$dbname:$hostname:3306";
    my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

    #Get all the MD5sums of the records in the database that are not closed
    my $SQLh=$itpslservh->prepare("SELECT md5sum FROM hwinfo WHERE dyear is NULL AND dmonth
is NULL");
    $SQLh->execute();
    my @md5hits;
    while (my $row=$SQLh->fetchrow_array()) {
        push(@md5hits, $row);
    #With this method we retrieve multiple results
    }

    #Now sample the current PCI activity and generate the live md5sums for each file record
    #First for PCI devices
    my $livepci=`$hwfilterstring1`;
    #And then for USB devices
    my $liveusb=`$hwfilterstring2`;
    my @livepcia=split("--\n",$livepci);
    my @liveusba=split("--\n",$liveusb);
    my @livemd5;

    foreach my $pcirec (@livepcia) {
        my ($dev1,$string1,$vendor,$string2)=split("\n",$pcirec);
        my @devvendor=split(":", $vendor);
        my @devstring1=split(":", $string1);
        my @devstring2=split(":", $string2);
        my $devstring=$devstring1[1].$devstring2[1];
        my $devbus="PCI";
        #Push the md5sum for the PCI device record in the livemd5 array
        push(@livemd5,md5_hex(join("," , $devvendor[1],$devstring,$devbus)));
    } #end of foreach loop

    foreach my $usbrec (@liveusba) {
        my ($garbage,$vendor,$string)=split("\n",$usbrec);
				my ($v1,$v2,$v3,$v4,$v5,$v6,$v7,$v8)=split(" ",$vendor);
				my $devvendor="$v3 "."$v4 "."$v5 "."$v6 "."$v7 "."$v8";
				my ($s1,$s2,$s3,$s4,$s5,$s6,$s7,$s8)=split(" ",$string);
				my $devstring="$s3 "."$s4 "."$s5 "."$s6 "."$s7 "."$s8";
				my $devbus="USB";
				#Push the md5sum for the USB device record in the livemd5 array
				push(@livemd5,md5_hex(join("," , $devvendor,$devstring,$devbus)));
		} #end of foreach loop

		#Contrary to the fact that we had to hit the database twice on dbupdate, when we check for existing records, 
we collected
		#all the PCI and USB MD5 sums and we need to perform the check only once this time
		#For each md5 record obtained from RDBMS, check if it exists in the live..
		#If it doesn't timestamp the closing time table columns for the record in the RDBMS.
		#To do that, first take the difference between the two arrays (the difference of the md5hits to the livemd5)
		my @arrunion=grep!${{map{$_,1}@livemd5}}{$_},@md5hits;

		foreach my $tobeclosed (@arrunion) {
				my ($dyear,$dmonth,$dday,$dhour,$dmin,$dsec)=timestamp();
				my $rows=$itpslservh->do ("UPDATE hwinfo set 
dyear='$dyear',dmonth='$dmonth',dday='$dday',dhour='$dhour',dmin='$dmin',dsec='$dsec'"
									. " where md5sum='$tobeclosed'" );
		}#end of foreach $tobeclosed

		$SQLh->finish();

} #end of chnonexistinghw


sub getdbauth {
		unless(open DBAUTH, "</root/itpsl/.adb.dat") {
						die "endpointresolver.pl Error:getdbauth: Could not open the .adb.dat file due to: 
$!";
				}

		my @localarray;

		while (<DBAUTH>) {
				my $dbentry=$_;
				chomp($dbentry);
				push(@localarray, $dbentry);
		}

		return @localarray;

} #end of getdbauth()
```

Appendix B : ITPSL sample source code

```
#!/usr/bin/perl

#getsig.pl - Retrieves one or more signatures from the ITPSL signature repository.
#George Magklaras - Center for Security, Commununications and Networks Research
#University of Plymouth, UK - February 2011
#Takes as arguments a space delimited list of ITPSL signid's

use DBI;
use strict;

my $VERSION="beta1";

#Some essential sanity checks
my @whoami=getpwuid($<);
die "getsig.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

my @arguments= @ARGV;

foreach (@arguments) {
		sigget($_);

}

sub sigget {
		#get the signid
		my $sigid=shift;
		#And here we connect to the database
		#get the db authentication info
		my @authinfo=getdbauth();
		my ($username,$dbname,$dbpass,$hostname);

		foreach my $dbentry (@authinfo) {
			($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
		}

		#Connect to the database
		my $datasource="DBI:mysql:$dbname:$hostname:3306";
		my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

		#Execute the statement and get the results by quoting the formed string
		#$sqlstring=$itpslservh->quote($sqlstring);
		my $SQLh=$itpslservh->prepare("SELECT mainblock FROM repository WHERE signid='$sigid' ");
		$SQLh->execute();

		print "Retrieving signature with id: $sigid from the ITPSL repository: \n";

		my @sighits=$SQLh->fetchrow_array();
```

```
        if (defined ($#sighits) && $#sighits=="0") {
                 open(my $fh, '>', "$sigid.xml") or die "getsig.pl Error:can't open the file to write signature
$sigid due to: $! \n";
                 select $fh;
                 print "$sighits[0]";
                 close $fh;
                 } else {
                print "getsig.pl Error: No signature entry for signature id:$sigid was found. \n"
        } #end of if

} #end of sigget


sub getdbauth {
        unless(open DBAUTH, "</root/itpsl/.sig.dat") {
                        die "getsig.pl Error:getdbauth: Could not open the .sig.dat file due to: $!";
                }

        my @localarray;

        while (<DBAUTH>) {
                my $dbentry=$_;
                chomp($dbentry);
                push(@localarray, $dbentry);
        }

        return @localarray;

} #end of getdbauth()
```

```
#searchsig.pl - Searches the ITPSL signature repository for signatures
#George Magklaras - Center for Security, Commununications and Networks Research
#University of Plymouth, UK - February 2011
#Takes a range of arguments to specify different search criteria according to the ITPSL ontology.
#The following search criteria are supported in the form of command-line switches
# --reason      VALUES: intentional/accidental
# --multihost   VALUES: yes/no
# --forhosts    VALUES: comma separated list of hostnames                    Example: host1,host2,host3
# --synopsis    VALUES: relational operators lists in the form OP=term1,term2 or single term    Example:
OR=development,production
# --keywords    VALUES: relational operators lists in the form OP=term1,term2 or single term    Example:
AND=p2p,azureus
# --context     VALUES: predictive/detection
# --fromdate    VALUES: dd/mm/yyyy                                   Example: 24/05/2010
# --todate      VALUES: dd/mm/yyyy                                  Example: 28/09/2010

use DBI;
use strict;
use Cwd;
use Getopt::Long;

my $VERSION="beta1";
my $reason;
my $multihost;
my $forhosts;
my $synopsis;
my $keywords;
my $context;
my $fromdate;
my $todate;
my $helpvar;
my $sqlstring;

GetOptions("reason=s" => \$reason,
                        "multihost=s" => \$multihost,
                        "forhosts=s" => \$forhosts,
                        "synopsis=s" => \$synopsis,
                        "keywords=s" => \$keywords,
                        "fromdate=s" => \$fromdate,
                        "context=s" => \$context,
                        "todate=s" => \$todate,
                        "help" => \$helpvar );

if ($helpvar) {
                dispusage();
}

#Some essential sanity checks
my @whoami=getpwuid($<);
die "submitsig.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

#Input validation checks
#If nothing is defined display by default the usage of the program
if ((!defined($reason)) && (!defined($multihost)) && (!defined($forhosts)) && (!defined($synopsis)) && (!
defined($keywords)) && (!defined($context)) && (!defined($fromdate)) && (!defined($todate)) ) {
            print "searchsig.pl Error: You provided no arguments \n";
            dispusage();
}

#Valid date formats
die "searchsig.pl Error: $fromdate is an invalid date format for a --fromdate argument. Use --fromdate dd/mm/yyyy .\n"
if ( !($fromdate =~ m/\d{2}\/\d{2}\/\d{4}/) && (defined($fromdate)) );
die "searchsig.pl Error: $todate is an invalid date format for a --todate argument. Use --todate dd/mm/yyyy .\n"
if ( !($todate =~ m/\d{2}\/\d{2}\/\d{4}/) && (defined($fromdate)) );

#Valid reason strings
die "searchsig.pl Error: $reason is an invalid string for a --reason argument. Use --reason intentional OR --reason
accidental .\n"
if ( defined($reason) && !($reason eq "intentional") && !($reason eq "accidental"));

#Valid multihost strings
die "searchsig.pl Error: $multihost is an invalid string for a --multihost argument. Use --multihost yes OR --multihost
no. \n"
if ( defined($multihost) && !($multihost eq "yes") && !($multihost eq "no") );

#Valid context strings
```

```
die "searchsig.pl Error: $context is an invalid string for a --context argument. Use --context predictive OR --context
detection .\n"
if ( defined($context) && !($context eq "predictive") && !($context eq "detection") );

#If one defines the fromdate switch there must be a todate switch as well.
die "searchsig.pl Error: When you define a --fromdate argument, you must also define a --todate argument. \n"
if ( defined($fromdate) && (!(defined($todate))));

#Equally for multihost 'yes' and --forhosts switch
die "searchsig.pl Error: When you define a --multihost yes argument, you also need to define a --forhosts argument. \n"
if ( defined($multihost) && $multihost eq "yes" && (!(defined($forhosts))) );

#For the synopsis switch
die "searchsig.pl Error: $synopsis is an invalid string for a --synopsis argument. Use something like --synopsis
AND=term1,term2  or --synopsis term1 \n"
if ( (defined($synopsis)) && !($synopsis =~ m/AND=((\w){2,},){1,}/) && !($synopsis =~ m/OR=((\w){2,},){1,}/)
        && !($synopsis =~ m/NOT=((\w){2,},){1,}/) && !($synopsis =~ m/(\w){2,}/) );

#For the keywords switch
die "searchsig.pl Error: $keywords is an invalid string for a --keywords argument. Use something like --keywords
AND=term1,term2 or --keywords term1 \n"
if ( (defined($keywords)) && !($keywords =~ m/AND=((\w){2,},){1,}/) && !($keywords =~ m/OR=((\w){2,},){1,}/)
        && !($keywords =~ m/NOT=((\w){2,},){1,}/) && !($keywords =~ m/(\w){2,}/) );

#We have done our basic checks so, we will start forming the SQL string
$sqlstring="SELECT signid from repository WHERE ";

#Here we transform the defined arguments into SQL statements.
#Their assembly to an SQL statements comes later on.
my $reasonsql;
if ( defined($reason)) {
        $reasonsql="reason='$reason'";
}

my $multihostsql;
if ( defined($multihost)) {
        $multihostsql="multihost='$multihost'";
}

my $forhostssql;
if ( defined($forhosts)) {
        if ($forhosts =~ m/((\w){1,},){1,}/) {
                my @hostarray=split("," , $forhosts);
                #Start the SQL string
                my $firstelement=shift @hostarray;
                $forhostssql="(hostlist RLIKE '$firstelement' ";
                foreach (@hostarray) {
                        $forhostssql=$forhostssql." OR hostlist RLIKE '$_' ";
                }
                #At the end, terminate the string
                $forhostssql=$forhostssql." )";
        } else {
                $forhostssql="hostlist RLIKE '$forhosts'";
        } #end of if-else

} #end of if ( defined($forhosts))

my $synopsissql;
```

```
if ( defined($synopsis)) {
        if ( $synopsis =~ m/AND=((\w){2,},){1,}/ || $synopsis =~ m/OR=((\w){2,},){1,}/) {
                #Get the operator and then the term list
                my ($synop,$therestof)=split("=", $synopsis);
                my @synarray=split(",", $therestof);
                my $firstelement=shift @synarray;
                $synopsissql="(synopsis RLIKE '$firstelement' ";
                foreach (@synarray) {
                        $synopsissql=$synopsissql." AND $synop synopsis RLIKE '$_' ";
                }
                #At the end, terminate the string
                $synopsissql=$synopsissql." )";
        } elsif ($synopsis =~ m/NOT=((\w){2,},){1,}/) {
                my ($synop,$therestof)=split("=", $synopsis);
                my @synarray=split(",", $therestof);
                my $firstelement=shift @synarray;
                $synopsissql="(NOT synopsis RLIKE '$firstelement' ";
                foreach (@synarray) {
                        $synopsissql=$synopsissql." AND $synop synopsis RLIKE '$_' ";
                }
                #At the end, terminate the string
                $synopsissql=$synopsissql." )";


        } else {
                $synopsissql="synopsis RLIKE '$synopsis'";
        } #end of if-else
} #end of if ( defined($synopsis))

my $keywordssql;
if ( defined($keywords)) {
        if ( $keywords =~ m/AND=((\w){2,},){1,}/ || $keywords =~ m/OR=((\w){2,},){1,}/) {
                #Get the operator and then the term list
                my ($keywop,$therestof)=split("=", $keywords);
                my @keywarray=split(",", $therestof);
                my $firstelement=shift @keywarray;
                $keywordssql="( keywords RLIKE '$firstelement' ";
                foreach (@keywarray) {
                        $keywordssql=$keywordssql." $keywop keywords RLIKE '$_' ";
                }
                #At the end, terminate the string
                $keywordssql=$keywordssql." )";
        } elsif ($keywords =~ m/NOT=((\w){2,},){1,}/) {
                #Get the operator and then the term list
                my ($keywop,$therestof)=split("=", $keywords);
                my @keywarray=split(",", $therestof);
                my $firstelement=shift @keywarray;
                $keywordssql="( NOT keywords RLIKE '$firstelement' ";
                foreach (@keywarray) {
                        $keywordssql=$keywordssql." AND $keywop keywords RLIKE '$_' ";
                }
                #At the end, terminate the string
                $keywordssql=$keywordssql." )";
        } else  {
                $keywordssql="keywords RLIKE '$keywords'";
        } #end of if-else
} #end of if ( defined($keywords))
```

```
my $contextsql;
if ( defined($context)) {
            if ($context eq "predictive") {
                    $contextsql=" NOT weightmatrix='0' "} else {
                    $contextsql=" weightmatrix='0' ";
                    }
}


my $fromdatesql;
if ( defined($fromdate) ) {
            my ($day,$month,$year)=split("/", $fromdate);
            $fromdatesql="year>='$year' AND month>='$month' AND day>='$day'";

}

my $todatesql;
if ( defined($todate) ) {
            my ($day,$month,$year)=split("/", $todate);
            $todatesql="year<='$year' AND month<='$month' AND day<='$day' ";
}

#Eventually, to form the SQL query string and place the AND operator to combine the
#criteria, we do the following.
#Define an array with all the defined arguments
my @definedargs;
foreach ($reasonsql, $multihostsql,$forhostssql,$synopsissql,$keywordssql,$contextsql,$fromdatesql,$todatesql) {
                if (defined($_)) {
                        push(@definedargs, $_);
                }
} #end of foreach

#Take the first argument statement that was defined
my $first=shift @definedargs;
$sqlstring=$sqlstring." $first";

#Then append the rest by means of AND ops (all the search criteria need to be valid)
foreach (@definedargs) {
                $sqlstring=$sqlstring." AND $_ ";
}


#Debug
print "Executing the query: $sqlstring \n";

#And here we connect to the database
#get the db authentication info
my @authinfo=getdbauth();
my ($username,$dbname,$dbpass,$hostname);

foreach my $dbentry (@authinfo) {
                ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
}

#Connect to the database
my $datasource="DBI:mysql:$dbname:$hostname:3306";
my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});
```

```
#Execute the statement and get the results by quoting the formed string
#$sqlstring=$itpslservh->quote($sqlstring);
my $SQLh=$itpslservh->prepare($sqlstring);
$SQLh->execute();

my @searchhits=$SQLh->fetchrow_array();

print "The ITPSL signatures that match are: \n";
foreach (@searchhits) {
                print "$_ \n";
}

#Subroutine definitions here
sub dispusage {
                print "Usage  :./searchsig.pl --reason REASON --multihost MLTHOST --forhosts HOSTS --synopsis
SEARCH_STRING --keywords KEYWRD_STRING --fromdate DATE --todate DATE [--help] \n";
                print "Example:./searchsig.pl --reason intentional --multihost yes --forhosts cn1,cn2 --synopsis 'p2p OR
ktorrent' --keywords 'theft' --fromdate 28/05/2010 --todate 29/05/2010 \n";
                exit;
} #end of dispusage

sub firsttimesigsub {
        print "searchsig.pl: It looks as if you are running the program for the first time. I will need some info from you
\n";
        print "searchsig.pl: Please enter the MySQL password for the root user and press Enter:";
        chomp (my $mysqlp=<STDIN>);
        print "searchsig.pl: Please verify the host name or IP of the LUARM RDBMS server:";
        chomp (my $luarms=<STDIN>);

        #Write the info into the config file
        open(my $fh, '>', "/root/itpsl/.sig.dat") or die $! or die "submitsig.pl Error:can't open the .sig.dat file: $! \n";
        select $fh;
        print "root,signatures,$mysqlp,$luarms";
        close($fh);
} #end of firsttimesigsub()

sub getdbauth {
                unless(open DBAUTH, "</root/itpsl/.sig.dat") {
                                                die "submitsig.pl Error:getdbauth: Could not open the .adb.dat file due to: $!";
                                }

                my @localarray;

                while (<DBAUTH>) {
                                my $dbentry=$_;
                                chomp($dbentry);
                                push(@localarray, $dbentry);
                }

                return @localarray;

} #end of getdbauth()
```

```
#!/usr/bin/perl
```

```
#submitsig.pl - Submits an ITPSL signature to the ITPSL signature repository
#George Magklaras - Center for Security, Commununications and Networks Research
#University of Plymouth, UK - February 2011
#Takes one argument, the filename of the ITPSL signature.
#It requires the presence of an XSD Schema file at the same directory.
#The program will validate the XML syntax of the submitted signature, check the consistency
#of various signature and if successful it will submit it to the ITPSL signature repository.
#Most of the error checking is done here. The itpslparser.pl handles only signatures that
#have successfully entered the database repository.

use XML::LibXML;
use XML::Twig::XPath;
use Class::Date;
use Digest::MD5 qw(md5 md5_hex md5_base64);
use Scalar::Util qw(looks_like_number);
use DBI;
use strict;
use Cwd;

my $VERSION="beta1";

#####################################
#IMPORTANT SIGNATURE DATA VARIABLES#
#####################################
#Signature date info and revision
my $cyear;
my $cmonth;
my $cday;
my $sigrev;
#The OS the signature is written for and its version
my $os;
my $osver;
#The user role the signature applies to
my $userrole;
#The detectby string (aka what do we use for detecting the threat)
my $detectm;
#The type of misuse (accidental or intentional)
my $reason;
#The host list that the signature is applicable to
my $hostlist;
#The Weight Matrix signature data
my $weightmatrix;
my $events;
#Multihost signature?
my $ismultihost;
#The threat keywords
my $keywords;
#Synopsis description (optional)
my $synopsis;
#and finally the body of the statements
my $itpslsig;


#Some essential sanity checks
my @whoami=getpwuid($<);
die "submitsig.pl Error:You should execute this program ONLY with root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);
```

```
my $filename=shift;
my $wdir=getcwd();

my $xmlschema = XML::LibXML::Schema->new( location => "$wdir/validate.xsd" );

#Does the signature file exist?
die "submitsig.pl Error: The signature file $filename does not exist. Are you sure you are referencing the file
properly? \n"
if (! (-e $filename));

#If it does, make a new LibXML object to parse it.
my $doc = XML::LibXML->new->parse_file($filename);

#Now check that the signature file has a valid ITPSL syntax
eval { $xmlschema->validate( $doc ); };
die "submitsig.pl Error: Oops! It seems that the signature $filename is not a valid one. Here is the error: $@ " if $@;

#Now that we have a valid signature, let's attempt to connect to the LUARM RDBMS server
die "submitsig.pl Error: First time run? The /root/itpsl directory does not exist. Please create it first and then re-run the
program.\n"
if (! (-e "/root/itpsl"));

#Do we reach the /root/itpsl/.sig.dat file?
if (! (-e "/root/itpsl/.sig.dat")) {
        firsttimesigsub();
}


#We start the parsing for checks here and we connect the different parts of the
#signature to the appropriate handling routines that will extract the value
#from the ITPSL markup.
my $twig = new XML::Twig::XPath( TwigHandlers => {
                #ITPSL header parsing data
                "/itpslsig/itpslheader/signdate/year" => \&getyear,
                "/itpslsig/itpslheader/signdate/month" => \&getmonth,
                "/itpslsig/itpslheader/signdate/day" => \&getday,
                "/itpslsig/itpslheader/ontology/revision" => \&getrev,
                "/itpslsig/itpslheader/ontology/weightmatrix" => \&getwm,
                "/itpslsig/itpslheader/ontology/detectby" => \&getdetectmethods,
                "/itpslsig/itpslheader/ontology/os" => \&getos,
                "/itpslsig/itpslheader/ontology/osver" => \&getosver,
                "/itpslsig/itpslheader/ontology/user_role" => \&geturole,
                "/itpslsig/itpslheader/ontology/reason" => \&getreason,
                "/itpslsig/itpslheader/ontology/multihost" => \&getmhost,
                "/itpslsig/itpslheader/ontology/hostlist" => \&gethlist,
                "/itpslsig/itpslheader/ontology/keywords" => \&getkeywords,
                "/itpslsig/itpslheader/ontology/synopsis" => \&getsynopsis,



});
# Now we have XML Schema validated let's parse, checking the logical validity of the
# parsed nodes on the way
$twig->parsefile( $filename );
$twig->flush;

#This is a new Twig object now to get the entire signature in $itpslsig
my $twig2=XML::Twig->new(pretty_print=> 'indented', 'xml:space'=>"preserve");
$twig2->parsefile( $filename );
```

```
$itpslsig=$twig2->sprint;


#At that point, all data should be fine and we should hit the LUARM signature repository

#Calculate the MD5 hex checksum of the signature by simply joining all the signature strings together
my $sigmd5=md5_hex(join("," , $cyear,$cmonth,$cday,$os,$osver,$sigrev,$reason,$weightmatrix,$detectm,$userrole,
$ismultihost,$hostlist,$keywords,$synopsis,$itpslsig));



#get the db authentication info
my @authinfo=getdbauth();
my ($username,$dbname,$dbpass,$hostname);

foreach my $dbentry (@authinfo) {
                ($username,$dbname,$dbpass,$hostname)=split("," , $dbentry);
}

#Connect to the database
my $datasource="DBI:mysql:$dbname:$hostname:3306";
my $itpslservh=DBI->connect ($datasource, $username, $dbpass, {RaiseError => 1, PrintError => 1});

#Check to see if the signature we are trying to submit already exists.
my $SQLh=$itpslservh->prepare("SELECT COUNT(*) FROM repository WHERE signid='$sigmd5' ");
$SQLh->execute();
my @md5hits=$SQLh->fetchrow_array();
if ($md5hits[0]=="1") {
                $SQLh->finish();
                die "submitsig.pl Error: The signature you tried to submit (MD5 hash $sigmd5) already exists in the ITPSL
signature repository. \n";
}

#If it doesn't exist INSERT the record
#by quoting parts that might contain characters that can force the execution of SQL statements to fail
#in the itpslsig, the keywords and synopsis fields
$itpslsig=$itpslservh->quote($itpslsig);
$synopsis=$itpslservh->quote($synopsis);
$keywords=$itpslservh->quote($keywords);

my $rows=$itpslservh->do ("INSERT INTO repository
(signid,year,month,day,reason,revision,userrole,detectby,multihost,hostlist,os,osversion,"
                                                            . "weightmatrix,keywords,synopsis,mainblock) "
                                                            . "VALUES
('$sigmd5','$cyear','$cmonth','$cday','$reason','$sigrev',"
                                                            .
"'$userrole','$detectm','$ismultihost','$hostlist','$os','$osver','$weightmatrix',$keywords,$synopsis,$itpslsig)" );

if (($rows==-1) || (!defined($rows))) {
                                        print "submitsig.pl Error: Could not insert signature with MD5 hash $sigmd5 into
the ITPSL repository. Something wrong occurred at the RDBMS level. \n";
                             }
#Finally close the db handler
$SQLh->finish();

print "submitsig.pl SUCCESS: Inserted signature with MD5 hash $sigmd5 into the ITPSL repository. \n";

#Subroutine definitions here
sub firsttimesigsub {
    print "submitsig.pl: It looks as if you are running the program for the first time. I will need some info from you
\n";
    print "submitsig.pl: Please enter the MySQL password for the root user and press Enter:";
    chomp (my $mysqlp=<STDIN>);
    print "submitsig.pl: Please verify the host name or IP of the LUARM RDBMS server:";
    chomp (my $luarms=<STDIN>);

    #Write the info into the config file
    open(my $fh, '>', "/root/itpsl/.sig.dat") or die $! or die "submitsig.pl Error:can't open the .sig.dat file: $! \n";
    select $fh;
    print "root,signatures,$mysqlp,$luarms";
    close($fh);
} #end of firsttimesigsub()

sub getdbauth {
        unless(open DBAUTH, "</root/itpsl/.sig.dat") {
                        die "submitsig.pl Error:getdbauth: Could not open the .adb.dat file due to: $!";
                }

        my @localarray;

        while (<DBAUTH>) {
                my $dbentry=$_;
                chomp($dbentry);
                push(@localarray, $dbentry);
        }

        return @localarray;

} #end of getdbauth()

sub getyear {
        my( $tree, $elem ) = @_;
        $cyear=$elem->text;

} #end of sub getyear

sub getmonth {
        my( $tree, $elem ) = @_;
        $cmonth=$elem->text;

} #end of sub getmonth

sub getday {
        my( $tree, $elem ) = @_;
        $cday=$elem->text;

} #end of sub getday

#Obtains and checks the validity of weight matrix data
sub getwm {
        my( $tree, $elem ) = @_;
        my $wm=$elem->text;
        #Filter out any white space data from the parsed signature wm string
        $wm=~ s/\s//g;
        my @wmvals;
        push (@wmvals, split(",", $wm));
        #set the global values of interest
        my $fnev=$wmvals[0];
        if (looks_like_number($fnev)) {;
                #Check that the number of specified ITPSL event weights is consistent
                die "submitsig.pl signature logic Error: HEADER SCOPE: You specified 0 events and you have additional weights in the signature. Please correct the wm and try again. \n"
                if ( $fnev eq "0" && $#wmvals gt "0" );
                die "submitsig.pl signature logic Error: HEADER SCOPE: You specified $fnev events, but your signature contains $#wmvals+1. Please correct the wm and try again. \n"
                if ( $fnev!=$#wmvals);
                #If all is well, export the value of w:m string
                $weightmatrix=$wm;
                } else {
                die "submitsig.pl signature logic Error: HEADER SCOPE:Got $fnev from the Weight Matrix as the number of events. This does not look like a number to me.";
                }
} #end of sub getwm

sub getdetectmethods {
        my( $tree, $elem ) = @_;
        $detectm=$elem->text;

} #end of sub getdetectmethods

sub getos {
        my( $tree, $elem ) = @_;
        $os=$elem->text;
} #end of sub getos

sub getosver {
        my ( $tree, $elem ) = @_;
        $osver=$elem->text;

} #end of sub getosver

sub getrev {
        my ( $tree, $elem) = @_;
        $sigrev=$elem->text;
} #end of sub getrev

sub geturole {
        my ( $tree, $elem) = @_;
        $userrole=$elem->text;

} #end of sub geturole

sub getreason {
        my ( $tree, $elem) = @_;
        $reason=$elem->text;

} #end of sub getreason

sub getmhost {
        my ( $tree, $elem) = @_;
        $ismultihost=$elem->text;

} #end of sub getmhost
```

```
sub gethlist {
        my ( $tree, $elem) = @_;
        $hostlist=$elem->text;

} #end of sub gethlist

sub getsynopsis {
        my( $tree, $elem ) = @_;
        $synopsis=$elem->text;

} #end of sub getsynopsis

sub getkeywords {
        my( $tree, $elem ) = @_;
        $keywords=$elem->text;

} #end of sub getkeywords
```

Appendix C : Publications of the research project

Some of the publications were edited and republished. Therefore, only the most relevant edition is included for reference.

Magklaras G., Furnell S., Papadaki M. (2011), "LUARM – An audit engine for insider misuse detection", 6th International Annual Workshop on Digital Forensics & Incident Analysis (WDFIA 2011)

Magklaras G., Furnell S. (2010), "Insider Threat Specification as a Threat Mitigation Technique", Book Chapter, Advances in Information Security, Vol 49: Title: Insider Threats in Cyber Security, Probst, Christian W.; Hunker, Jeffrey; Gollmann, Dieter (Eds.) 2010, XII, 244 p. 40 illus., 20 in color., ISBN 978-1-4419-7132-6, Hardcover, Springer 2010.

Magklaras G., Furnell S., Brooke P. (2006), "Towards an Insider Threat Prediction Specification Language",Information Management & Computer Security, vol. 14, no. 4, pp. 361-381.

LUARM – An audit engine for insider misuse detection

G.Magklaras, S.Furnell and M.Papadaki

Centre for Security, Communications and Network Research, University of Plymouth, Plymouth, UK
cscan@plymouth.ac.uk

Abstract

'Logging User Actions in Relational Mode' (LUARM) is an open source audit engine for Linux. It provides a near real-time snapshot of a number of user action data such as file access, program execution and network endpoint user activities, all organized in easily searchable relational tables. LUARM attempts to solve two fundamental problems of the insider IT misuse domain. The first concerns the lack of insider misuse case data repositories that could be used by post-case forensic examiners to aid an incident investigation. The second problem relates to how information security researchers can enhance their ability to specify accurately insider threats at system level. This paper presents LUARM's design perspectives and a 'post mortem' case study of an insider IT misuse incident. The results show that the prototype audit engine has a good potential to provide a valuable insight into the way insider IT misuse incidents manifest on IT systems and can be a valuable complement to forensic investigators of IT misuse incidents.

Keywords

Insiders, misuse, detection, auditing, logging, forensics

1. Introduction

The problem of insider IT misuse is a very real threat for the health of IT infrastructures encompassing both intentional activities (e.g. targeted information theft and accidental misuse (e.g. unintentional information leak). Numerous studies have tried to define an "insider" in the context of Information Security. A generic definition from Probst et al. (2009) is "a person that has been legitimately empowered with the right to access, represent, or decide about one or more assets of the organization's structure".

The most widely known insider misuse cases are usually about intellectual property theft. The arrest of Lan Lee and Yuefei Ge by FBI agents (Cha, 2008) is a classic case. The arrested men were engineers of NetLogic Microsystems (NLM) until July 2003. During the time of their employment, they were downloading trade sensitive documents from the NLM headquarters into their home computers. These documents contained detailed descriptions of the NLM microprocessor product line. Eventually, their ties to the Chinese government and military were discovered by investigators. However, both mass media case descriptions and relevant security surveys do not provide the tools or the methodology to systemically study and mitigate the problem. Insider IT misuse is a multi-faceted problem and one of the things insider misuse researchers really need is a repository of more detailed case descriptions with a focus on the impact insider misuse actions have at computer system level (NSTISSAM). This is the area of Insider Threat Specification, the core concept behind the proposed logging engine which is examined in the next section.

2. Insider Threat Specification and modelling

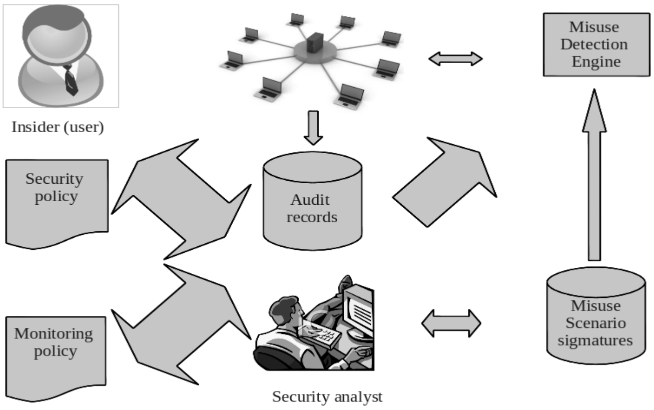


Figure 1: Information flow in a misuse detection system

Threat specifications follow the principles of intrusion specification, a concept which is not new in the information security world. Techniques to describe threats exist for an entire range of information security products, from anti-virus software to several intrusion detection/prevention systems (IDS/IPS) (Bace, 2000), where threats are specified by anomaly detection, pattern matching (also known as misuse detection) mechanisms or a heuristic-based combination of the two. Insider Threat Specification is the process of using a standardized vocabulary to describe in an abstract way how the aspects and behaviour of an insider relate to a security policy defined misuse scenario. Figure 1 shows the information flow of a typical IT misuse detection system. The security specialist translates the Security (and resulting monitoring policy) into a set of misuse scenario signatures, standard descriptions of IT misuse acts that describe the behaviour of a user at process execution, filesystem and network endpoint level (Magklaras et al, 2006). The misuse scenario signatures and collected audit data (Bace, 2000) from the IT infrastructure are fed into a misuse detection engine.

Vital to insider threat specification is the structure and content of the audit record, at the center of Figure 1. If the audit record is incomplete, in terms of the type of information we need to log or unavailable, because the data are vanished due to bad system design or intentional data corruption, the specification of insider threats is useless. This is one of the primary objectives that LUARM tries to address by providing an evidence rich and reliable audit record format.

3. Insider misuse detection auditing requirements

Bace (Bace, 2000) discusses intrusion detection (and hence misuse detection) as an audit reduction problem. Audit reduction is the process of filtering the relevant information out of the audit records, in order to infer a partially or fully realized threat and excluding information that is irrelevant or redundant. The structure of an audit record is important for a misuse detection system. A good structure has well defined fields that can be easily parsed. Moreover, the structure of the audit record should easily facilitate relational type queries. It is necessary for the information to be applied on the disjunction (OR), conjunction (AND), and negation (NOT) operators, in order to increase the query versatility and speed of response.

A desired aspect of a suitable crafted audit record format for insider misuse detection is clear user accountability. This means that the audit record should be able to reliably and easily associate user entities to recorded actions. The wealth and replication of vital information in various types of audit records is a requirement for proper event correlation and step instance selection (Meier, 2004).

Another important issue of audit record engines is that of referencing time. In large IT infrastructures that span several networks and time zones, audited systems might report in different time formats. They can also experience 'clock skew', a difference in time recorded amongst computer systems due to computer clock hardware inaccuracies, especially when an NTP (Mills et al, 2010) server is not available to provide a reliable time source.

One of the most recent and commonly referenced works that concern the format of audit records is the Common Criteria for Information Technology Security Evaluation (Common Criteria Portal, 2009) standards. The Common

Criteria (CC) effort does not fully address the previously mentioned audit record requirement omissions of its predecessor, the Orange Book (DOD 5200.28-std, 1985). However, some of its high level functional audit requirements are interesting. In particular, CC requirement 88 of section 8.2 states that: “At FAU_GEN.2 User identity association, the TSF shall associate auditable events to individual user identities.” In CC terminology TSF stands for Target of evaluation Security Functionality, meaning essentially the software and hardware under evaluation. In addition, CC mentions a set of requirements that concern various aspects of the audit record storage. Once again, the requirements are given in high-level terms, specifying that:

- unauthorized deletion and/or modification of audit records
- any other condition that could cause storage failure.

should be mitigated.
The next section discusses whether today's audit engines satisfy these requirements.

4. Existing audit record engines

Audit record engines have existed since the very early days of operating systems. However, not all of them fit the requirements of misuse detection engines, as discussed in the previous section.

The most common variety of audit record engines uses information that comes directly from the Operating System. Characteristic examples of this category of engines are Oracle's Basic Security Module (BSM) auditing system (Oracle Corporation, 2010) and its open source implementation OpenBSM (Trusted BSD Project portal, 2009), the psacct audit package (psacct utilities, 2003), as well as the syslogd (Gerhards, 2009) and WinSyslogd (Monitorware, 2010) applications.

After examining these engines, serious deficiencies can be located in terms of use for insider threat prediction. Firstly, many engines consolidate information from various different devices and operating system vendors, but they are far from describing sufficiently issues in an operating system agnostic way. In addition, process accounting tools might not cover sufficiently the variety of different system level information (file, process execution and network level). In fact, some of them might miss data as described in (HP Portal, 2003). A logging engine that cannot facilitate the description of both static and live forensic insider misuse system data at the network, process and filesystem layer could hinder a forensic examination of an IT misuse incident. Static digital forensic analysis is employed by most forensic tools and cannot portray accurately the non-quiescent (dynamic) state of the system under investigation. Information such as active network endpoints, running processes, user interaction data (number of open applications per user, exact commands), as well as the content of memory resident processes may not be recorded accurately on non-volatile media. (Hay et al, 2009) discuss the shortcomings of static digital forensics analysis in detail. In order to overcome the barriers of static analysis, Adelstein et al. (2006) discuss the virtues of non-quiescent or live analysis, which essentially gathers data while the system under-investigation is operational.

Several audit record systems do not report consistently the timing of audit record generation. For instance, many implementations of the syslog audit standard and psacct tools generate the audit record by entering the time stamp of the client system. If the client system does not have a reliable time source, this generates inaccurate information and could seriously hinder event correlation.

Finally, one of the most serious drawbacks of existing audit approaches is the inability to store the audit information in a form that can utilize relational queries. Section 3 discussed the reasoning behind this requirement. In one sense, some people might argue that this is an audit management feature rather than an audit log design issue. However, as section 3 discussed the advantages of using a relational schema to form audit queries in a structured log record, the author's view is that everything that increases the expressive power of an audit log query should be incorporated in the structure of the audit log, rather than being left as an 'add-on' feature.

5. The LUARM audit engine

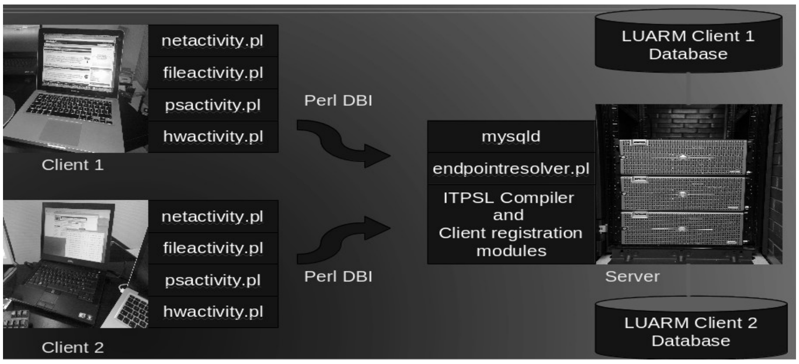
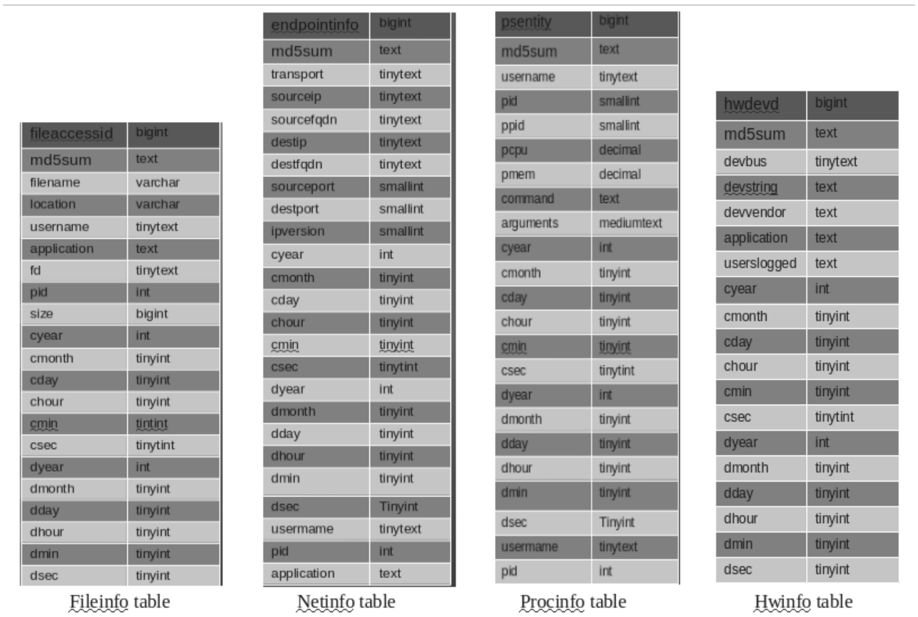


Figure 2: The LUARM architecture

LUARM is a prototype Open Source audit record engine (LUARM portal, 2010) that uses a Relational Database Management System (RDBMS) for the storage and organization of audit record data. The employment of an RDBMS is a core design choice for the LUARM engine. Beyond the relational type query support discussed in Section 3, an RDBMS offers the necessary data availability, integrity and scalability features, because most RDBMS tools are explicitly designed to organize and store large amounts of data, as dictated by many CC requirements. The Structured Query Language (SQL) facilitates instance selection and completion, as well as data correlation can be performed by using clauses such as 'FROM' and 'WHERE'.



fileaccessid	bigint
md5sum	text
filename	varchar
location	varchar
username	tinytext
application	text
fd	tinytext
pid	int
size	bigint
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tintint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	tinyint

Fileinfo table

endpointinfo	bigint
md5sum	text
transport	tinytext
sourceip	tinytext
sourcefqdn	tinytext
destip	tinytext
destfqdn	tinytext
sourceport	smallint
destport	smallint
ipversion	smallint
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	Tinyint
usermame	tinytext
pid	int
application	text

Netinfo table

psentity	bigint
md5sum	text
username	tinytext
pid	smallint
ppid	smallint
pcpu	decimal
pmem	decimal
command	text
arguments	mediumtext
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	Tinyint
usermame	tinytext
pid	int

Procinfo table

hwdevd	bigint
md5sum	text
devbus	tinytext
devstring	text
devvendor	text
application	text
userslogged	text
cyear	int
cmonth	tinyint
cday	tinyint
chour	tinyint
cmin	tinyint
csec	tinytint
dyear	int
dmonth	tinyint
dday	tinyint
dhour	tinyint
dmin	tinyint
dsec	tinyint

Hwinfo table

Figure 3: LUARM relational table structure

Figure 2 depicts the module client-server architecture of the LUARM audit engine. On the left of the figure, we can see a set of audited computer clients. Every client is running a unique instance of a set of monitoring scripts. Each of the client scripts audits a particular system level aspect of the operating system: 'netactivity.pl' audits the addition and creation of endpoints, 'fileactivity.pl' records various file operations, 'psactivity' provides process execution audit records and 'hwactivity.pl' keeps a log of hardware devices that are connected or disconnected from the system. The right hand side contains the centralized server part of the architecture where audit data are stored, maintained and queried in a MySQL (Oracle MySQL portal, 2010) based RDBMS (other RDBMS systems could be used as well). The Perl programming language is used to implement the modules and the communication between client and server is performed via a Perl DBI (CPAN-DBI, 2010) interface.

The client-server architecture avoids leaving the data in vulnerable clients. The central host MySQL server has its own authentication system responsible for controlling who has access to the audit data. By authenticating audit reviewers against the RDBMS authentication system, we de-couple the users being audited from the auditors, a desirable property that ensures that audited insiders cannot easily manipulate audit data. Furthermore, by assigning a separate database instance per audited client, we reduce the likelihood of compromising the data for all clients. If the database access credentials of one client are compromised, the damage is limited to the audit data for that client only.

Figure 3 displays the relational table format for the four main types of recorded audit data in LUARM: fileaccess, process execution, network endpoint and hardware device information. Temporal information is provided by event creation time stamps (cyear, cmonth, cday,chour,cmin,csec) and respective event destruction time stamps (dyear,dmonth,dday,dhour,dmin,dsec). The combination of the two types of timestamps can pinpoint exact time intervals for events in a consistent format for all recorded event types. In contrast, most audit systems may provide only event creation time references without hinting for the duration of an event.

The sampling of events is done at 100ms intervals and is adjustable by means of modifying certain variables on each monitoring module. At first, this might seem problematic as many attack steps can occur much faster than that amount of time. However, in an event sampling loop, one has to account for the time delay to update the database, which can vary from 10ms to 60-70 ms intervals on heavily loaded clients and servers. In addition, time resolution varies amongst operating systems. For these reasons, LUARM relies on the Perl Time::HiRes module (CPAN-HiRes, 2010) to bridge the gap between the different operating system timer implementations. A time granularity of 100 ms is also a good compromise between accuracy and scalability. The more granular the time resolution, the greater the computational load for both the client and the server LUARM parts.

Another important design decision that concerns the format of the audit table was to include common attributes amongst different event tables for the purposes of increasing the ability to correlate events and provide user entity accountability. For instance, fields such as 'username' (user entity), pid (numeric process ID of the program responsible for the event creation) and application (string that represents the name of the application that matches the pid) can be found in most of the event tables. This enables the audit reviewer to use SQL and relate events, so he can form queries of the type “Find the network endpoint created by program x of user y” in an easy manner.

The 'fileinfo' table stores file access related events. The filename specification consists of two parts. The 'filename' field which holds the filename with the file extension (i.e. data.txt) and the 'location' field which contains the absolute path of the file. The fact that the two are divided in separate fields makes it easier to search by location or by field name only, increasing the versatility of mining file data. In order to populate the data on this table, LUARM relies on the 'lsof' utility (Pogue et al, 2008). The utility is versatile and can record a variety of events including file and network endpoints in real time. It exists for an entire range of UNIX/Linux and MACOSX operating systems, covering a large spectrum of computing devices.

The 'netinfo' table logs the creation and destruction of network endpoints. In the context of LUARM, the term 'network endpoint' refers to the operating system data structures employed to facilitate network connectivity via the TCP/IP protocol suite. Network endpoint activity is considered as live forensic data. A series of table fields are used to record endpoint details ('sourceip', 'destip', 'sourceport' , 'destport' and 'transport' record source and destination IP addresses, source and destination port and transport protocol respectively). The fields 'sourcefqdn' and 'destfqdn' hold the DNS (Mockapetris, 1987) resolved Fully Qualified Domain Name (FQDN) for the source and destination hosts.

The 'sourcefqdn' and 'destfqdn' fields are not populated by the client LUARM routines. In contrast, they are populated on the LUARM server side. Due to the criticality of correct DNS data for the audit records, the frequent DNS configuration errors (Barr, 1996), aspects of DNS operational security (Bauer, 2003) and client performance, the endpoint name resolution is left on the server side. This provides a greater control on DNS derived data and does not rely on vulnerable clients (malicious insiders or software vulnerabilities) for auditing network connections.

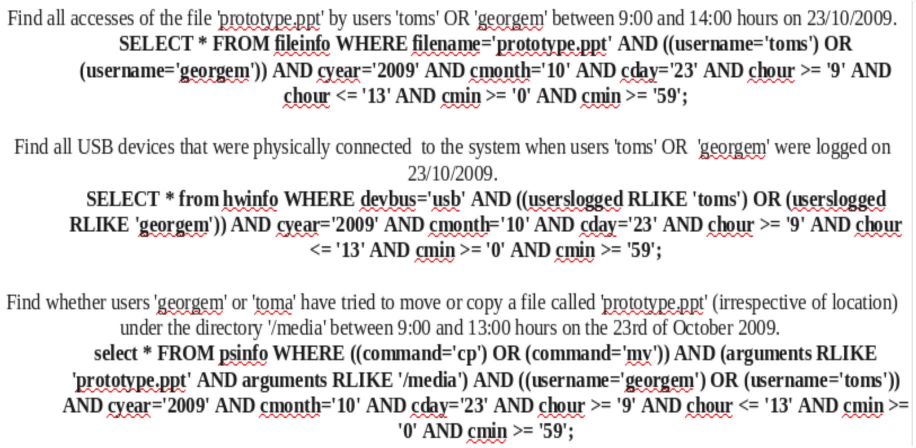
Process execution activity is recorded in the 'psinfo' table (Figure 3). This table records 'live' forensic data. The table includes both the proces ID ('pid') and parent process id ('ppid'), so that process execution flow can be traced back to the original process. In order to speed up process execution searches, the LUARM engine also separates the executed command ('command') from its arguments ('arguments'). One might like to search them separately in the process of mining process execution data. The 'ps' UNIX/Linux utility (Pogue et al, 2008) is used to collect process information. For all active processes (whose d* temporal fields are NULL), LUARM updates in near real time these two fields.

The 'hwinfo' table logs 'live' device connection and disconnection events. All events generated by devices that connect to the Peripheral Component Interconnect (PCI and PCI-Express) and Universal Serial (USB) buses. These two buses are commonly found on a large array of computing devices. For instance, an audit reviewer or forensics analyst might correlate file activity to a portable storage medium connection, as part of an intellectual property theft scenario. In that case, the 'hwinfo' table logs information in various fields that help identify the attached device ('devstring', 'devvendor'), the bus the device was connected to ('bus') and correlate the device attachment event against a number of users that are logged into the system at the time of the device attachment ('userslogged').

6. LUARM in action

Having a proposed structure and content for the various categories of audit events as described in the previous section, we can now issue sample SQL statements to illustrate how audit data mining is performed. Figure 4 displays sample queries that demonstrate the expressiveness of LUARM's audit record content and structure.

There are a few important observations to make about the example LUARM SQL queries. The first one concerns the embedding of system specific knowledge inside the statement. In essence, the third example of Figure 4 defines a step of an insider trying to transfer a sensitive file to a portable medium. One has to know the name of the sensitive file 'prototype.ppt' and also the fact that '/media' is used as a mount point for portable media for that host. Additional possible destination locations could be specified by means of OR operators. The use of the 'RLIKE' operator (RLIKE RegExp, 2008), always in relation to the second and third examples of Figure 4. The operator implements a regular expression type of match. Apart from the conjunction operator (OR), regular expressions give the specification polymorphic properties (one specification string, many matching results), a desirable property for compact misuse detection language statements.



Find all accesses of the file 'prototype.ppt' by users 'toms' OR 'georgem' between 9:00 and 14:00 hours on 23/10/2009.

```
SELECT * FROM fileinfo WHERE filename='prototype.ppt' AND ((username='toms') OR
(username='georgem')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND
chour <= '13' AND cmin >= '0' AND cmin >= '59';
```

Find all USB devices that were physically connected to the system when users 'toms' OR 'georgem' were logged on 23/10/2009.

```
SELECT * from hwinfo WHERE devbus='usb' AND ((userslogged RLIKE 'toms') OR (userslogged
RLIKE 'georgem')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour
<= '13' AND cmin >= '0' AND cmin >= '59';
```

Find whether users 'georgem' or 'toma' have tried to move or copy a file called 'prototype.ppt' (irrespective of location) under the directory '/media' between 9:00 and 13:00 hours on the 23rd of October 2009.

```
select * FROM psinfo WHERE ((command='cp') OR (command='mv')) AND (arguments RLIKE
'prototype.ppt' AND arguments RLIKE '/media') AND ((username='georgem') OR (username='toms'))
AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour <= '13' AND cmin >=
'0' AND cmin >= '59';
```

Figure 4: Using SQL to mine data in LUARM

LUARM was tested on a variety of simulated insider misuse scenarios. The scenarios were derived by real world LUARM captured data. However, permission to publish the original audit data was not obtained by the organizations in question. Thus, we had to reconstruct the misuse incidents by means of writing down a text based description of each incident and ask a team of users to re-enact it under a controlled IT infrastructure. The following paragraphs will present one of these incidents and demonstrate how the correlation versatility of the LUARM relational audit log structure can shed forensic light into the actions of a malicious insider. The scenario is provided below:

> 'Autobrake' Corp is a company designing car braking systems. Their engineering department is the most information sensitive work area. The braking system design process takes place in high performance Linux workstations, one for each design engineer. The engineers have normal user rights to the workstations. Superuser rights (root) is given only to the IT admin. The designs reside on the local hard drives of the workstations and the company's IT policy forbids any transfer of sensitive data to portable media. Autobrake's system administrator has requested a salary raise various times. This has been denied by management. The system administrator is lured by a competing company that asked him to deliver schematics of the new and revolutionary Autobrake's RGX9 SUV braking system in return for a large amount of money. Enjoying the trust of everyone and having full control of the engineering CAD workstations, the system administrator decides to take the offer of the competing company. He performs the intellectual property theft by following a well designed approach which is summarized below:

- He carefully chooses the user account of a mechanical engineer (username 'engineer3') that had some disputes over work issues with management. He aims to avoid detection by means of masquerading as the engineer in question.
- After successfully masquerading as the engineer in the IT system he uses a portable USB key to obtain the commercially sensitive RGX9 schematic, leaving only the traces of the engineer “actions”.

Assuming that a third party auditor manages the audit process and monitors the logging (ensuring that the logging infrastructure works) and that all Engineering workstations are monitored by LUARM, we are now tasked to find the offender and clear the name of 'engineer3'. The reader should consult the LUARM relational table structure (Figure 3), in order to follow the SQL queries presented below.

The investigation begins from the most important file, that of RGX9, and the people that work on it. From the audit record of the workstations with name 'proteas', we utilize LUARM to find out who has been using the file:

```
mysql> select username,pid,cday,chour,cmin,location,filename from fileinfo where filename RLIKE 'RGX9' OR location RLIKE 'RGX9' \G
```

From the many hits we get from the data base, we focus our attention on the following ones:

```
*************************** 111. row ***************************
username: engineer3
pid : 8301
cday: 4
chour: 15
cmin: 30
location: /storage/users/engineer3/work/designs
filename:RGX9.jpg
```

```
...
*************************** 118. row ***************************
username: engineer3
pid: 28538
cday: 4
chour: 15
cmin: 32
location: /media/U3SAN03-12
filename: RGX9.jpg
```

The reason these file access patterns looked suspicious is that they were different than the normal pattern of accessing the file by the staff engineer. Normally, user 'engineer3' would access the file by means of certain design and image editing applications, under its usual directory (/storage/users/engineer3/work/designs). This time, however, things look a bit different, if one follows the association of file access to process execution, in order to confirm which programs performed the file transaction. The following SQL queries achieve the desired association:

```
mysql>select username,pid,command,arguments,cyear,cday,chour,cmin from psinfo where
username='engineer3' AND pid='8031' AND cyear='2011' AND cday='4' AND chour='15' AND cmin='30;

*************************** 1. row ***************************
username: engineer3
pid: 8031
command: /bin/cp
arguments: work/designs/RGX9.jpg /tmp/
cyear: 2011
cday: 4
chour: 15
cmin: 30

mysql>select username,pid,command,arguments,cyear,cday,chour,cmin from psinfo where
username='engineer3' AND pid='8031' AND cyear='2011' AND cday='4' AND chour='15' AND cmin='30;

*************************** 1. row ***************************
username: root
pid: 28538
command: mv
arguments: RGX9.jpg /media/U3SAN03-12
cyear: 2011
cday: 4
chour: 15
cmin: 32
```

Essentially, the previous results verify that the file was first copied from the normal directory to /tmp and then was moved to the /mnt/usb. At this point, a little bit of system specific knowledge comes into light, as /mnt/usb is the usual mount point where Linux links portable storage media to the filesystem. Hence, the question to raise is whether a portal storage medium was connected to the workstation, prior to the 'mv' file transaction. The query result yields a positive answer:

```
mysql> select * from hwinfo where cyear='2011' AND cmonth='01' AND cday='04' AND chour='15'\G
*************************** 1. row ***************************
hwdevid: 71
md5sum: a16e7386f14de769a7a9491da2071f5b
cyear: 2010
cmonth: 12
cday: 4
chour: 15
cmin: 30
csec: 28
devbus: USB
```

```
devstring: Cruzer Micro U3
devvendor: SanDisk Corp.
userslogged: engineer3,root
dyear: 2010
dmonth: 1
dday: 4
dhour: 15
dmin: 33
dsec: 38
```

This database hit seems to be in line with the actions of engineer3, as it indicates a device connection before the execution of the 'mv' command and a disconnection well after the mv command. Thus, everything seems to point out that 'engineer3' violated the company policy and transferred a sensitive file to a USB medium, against the company IT regulations. However, this had been categorically denied by the actual person. A good but non IT based alibi for the staff engineer was that he exited the building with his security card token around 14:50, returning back to his desk at 15:50, a wide gap for him. Clearly, something else was going on and the clue was the 'userslogged' field of the last LUARM result. This 'hwinfo' LUARM table field contains the usernames for accounts that are logged into the workstation at the time of the device connection. Apart from 'engineer3' we note the root account being active, which is clearly the only other choice that, under the circumstances, could have performed the mount procedure.

Based on the time stamp of the mv operation, a careful investigation of the root account actions reveals a key command execution, derived from the 'psinfo' table:

```
mysql> select * from psinfo where pid='27865' AND cyear='2011' AND cday='4' AND cmonth='1' AND
chour='15' AND cmin >= '20' AND cmin <='33' \G
*************************** 1. row ***************************
psentity: 97654
md5sum: 7067284f2e1aefc430339ef091b4e41b
username: root
pid: 27865
ppid: 26407
pcpu: 0.0
pmem: 0.0
command: su
arguments: - engineer3
cyear: 2011
cmonth: 1
cday: 4
cmin: 28
chour: 15
csec: 36
dyear: 2011
dmonth: 1
dday: 4
dhour: 15
dmin: 28
dsec: 39
```

The 'su' command is used routinely by administrators to switch user credentials, in order to test environment settings and perform system tasks (Garfinkel et al, 1996). However, it can be easily used as a masquerading tool to covertly perform actions using the credentials of somebody else. A further investigation also found the USB key on the desk of the IT administrator with the RGX9.jpg file. The hwinfo table device identifier data ('devstring', 'devvendor') as well as the mount point identifier (/media/U3SAN03-12) from the psinfo commands contributed towards strengthening the final piece of the puzzle.

This case shows the versatility of the relational structure of the LUARM record that showed the way from simple file operation to related program execution and other events that can provide strong evidence and lead to the misuser. In addition, LUARM has also been used successfully to provide evidence about security incidents of external origin (Magklaras, 2011). Thus, it offers a valuable complement of existing logging mechanisms.

7. Conclusions

A very important tool to mitigate Insider IT misuse is an audit record which is specifically designed to address its various needs, as well as complement existing forensic tools when security specialists perform a post-mortem incident examination. LUARM is an audit engine that provides a detailed log of user actions at file, process execution and network endpoint level stored in a Relational Database Management System. Its file, process and network endpoint data provide a dynamic forensic view of the system, a useful complement to existing forensic tools that offer only static data in their majority. The relational storage layer increases the correlation versatility amongst the different types of audit data, as it is vital to be able to perform various associations during the investigation of an incident (process to file, process to network activity) and reliably relate actions to user entities.

The results are promising, showing a much better way to examine a system than looking at static text files which are difficult to parse and even more difficult to correlate. However, LUARM is a work in progress. It has its deficiencies and needs many improvements, in order to become a production real-world audit engine for insider misuse.

The first issue that was identified relates to the sampling frequency of user processes execution. After examining carefully the consistency of audit logs, it became evident that LUARM was losing process execution data. A fault was located at the process execution monitoring module. Due to the way the sampling loop was written in that module, the effective sampling frequency could exceed by far the desired 100 millisecond sampling frequency. As a result, LUARM would miss processes that executed by various users in the system. The module was re-written using an entirely different process execution sampling philosophy. A Linux kernel technique called 'execve wrapping' was employed by adopting the Snoopy logger open source software (Snoopylogger portal, 2000). A modified 'execve wrapper' logger like 'Snoopy' logger provides a way to log the process execution and its arguments without relying on a sampling loop and is thus a more efficient interface to capture live process execution data. This solved the problem of losing process execution data due to a slow sampling rate and thus corrected an important deficiency of LUARM.

Addressing the issue of user privacy is not so straightforward. There is always a tension between insider IT misuse monitoring and privacy. LUARM needs to retain and collect data about a user's behavior, in order to help the analyst infer IT misuse. In direct contrast, privacy dictates the right of individuals to define whether somebody will collect data about their online actions and the extent or way the data can be used. The best compromise between these two opposing needs is to control the amount and type of logged data. This can be achieved by pseudo-anonymizing certain parts of the audit record, in order to protect certain aspects of the user privacy but still be able to infer IT misuse reliably. The term 'Privacy-Respecting Intrusion Detection' (Flegel, 2007), encompasses all the efforts of achieving a good compromise between the need to monitor and the need to respect user privacy.

The achievement of the LUARM prototype has been to demonstrate that structured evidence based logging for IT misuse is feasible. The authors welcome feedback and participation to the development of its code base. The prototype is not yet ready for production deployment, but it should be suitable for experimentation and has already proved its value on a number of insider IT misuse incidents.

8. Acknowledgements

The authors wish to thanks the University of Oslo IT engineers Harald Dahle, Jean Lorentzen and Melaku Tadesse for helping with the simulation of various misuse scenarios.

9. References

Adelstein F. (2006), *"Live Forensics: Diagnosing Your System without Killing it First"*, *Comm. ACM*, vol.49, no.2, 2006, pp. 63-66.

Bace R. (2000), "*Intrusion Detection*", *Macmillan Technical Publishing*, Indianapolis, USA, ISBN: 1-578701856, pp. 38-39 discuss the terms 'misuse detection' and 'anomaly detection' in an intrusion specification context, pp. 47-66 discuss various audit record issues.

Barr D. (1996), "*Common DNS Operational and Configuration Errors*", *Internet Engineering Task Force (IETF) Request For Comment (RFC) 1537*, February 1996.

Bauer M. (2003), *"Building secure servers with Linux"*, *O'Reilly & Associates*, ISBN: 0-596-00217-3: Chapter 6, pp. 154-196.

Cha A.E. (2008), "*Even spies embrace China's free market.*", WashingtonPost,February15,2008,www.washingtonpost.com/wpdyn/content/article/2008/02/14/AR2008021403550.html (Accessed 03 March 2011)

Common Criterial Portal (2009), "*The Common Criteria for Information Technology Security Evaluation*", Version 3.1, Revision 3, July 2009. Part 2: Functional security components,www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R3.pdf (Accessed 03 March 2011)

CPAN-DBI (2010), "*The Perl Database Interface (DBI) module*" at the Comprehensive Perl Archive Network (CPAN),search.cpan.org/~timb/DBI-1.615/DBI.pm (Accessed 03 March 2011)

CPAN-HiRes (2010), "*The Perl High Resolution Timer module*" at the Comprehensive Perl Archive Network (CPAN),search.cpan.org/~jhi/Time-HiRes-1.9721/HiRes.pm (Accessed 03 March 2011)

DOD 5200.28-std (1985), "*Department of Defense Trusted Computer System Evaluation Criteria*", National Computer Security Center: Orange Book, DOD 5200.28-std, December 1985.

Flegel U. (2007), "Privacy-Respecting Intrusion Detection", Advances in Information Security, Springer, ISBN: 978-0-0387-34346-4 .

Furnell S. (2004), "*Enemies within: the problem of insider attacks*", *Computer Fraud and Security*, Volume 2004 Issue 7, pp. 6-11.

Garfinkel S, Spafford G. (1996), "*Practical UNIX and Internet Security*", Second Edition, O'Reilly and Associates, Sebastopol, CA, ISBN: 1-56592-148-1

Gerhards R. (2009), "*The Syslog Protocol*", Internet Engineering Task Force (IETF), Request for Comment (RFC) 5424, March 2009.

Hay B., Nance K., Bishop M. (2009), "*Live Analysis Progress and Challenges*", IEEE Security & Privacy, Volume 7, Number 2, pp. 30-37.

HP Portal (2003), "*psacct process accounting misses some commands*", HP IT ,forums11.itrc.hp.com/service/forums/questionanswer.doadmit=109447626+1286381845785+28353475&threadId=1413576 (Accessed 02 February 2011)

LUARM portal (2010),luarm.sourceforge.net/ (Accessed 03 March 2011)

Snoopylogger (2000), http://sourceforge.net/projects/snoopylogger/ (Accessed 04 May 2011)

Magklaras G., Furnell S., Brooke P. (2006), "*Towards an Insider Threat Prediction Specification Language*", Information Management & Computer Security, (2006) vol. 14, no. 4, pp. 361-381.

Magklaras G. (2011), "*Catching an undesired guest in the penguin /tmp room*", Epistolatory Blogspot, epistolatory.blogspot.com/2011/02/catching-undesired-guest-in-penguin-tmp.html (Accessed 03 March 2011)

Meier M. (2004), "A Model for the Semantics of Attack Signatures in Misuse Detection Systems", K. Zhang and Y. Zheng (Eds.): ISC 2004, Springer-Verlag Berlin,Heidelberg , LNCS 3225, pp. 158–169.

Mills D., Delaware U., Martin J., Burbank J., Kasch W. (2010), "*Network Time Protocol Version 4: Protocol and Algorithms Specification*", Internet Engineering Task Force (IETF) Request For Comment (RFC) 5905, June 2010.

Mockapetris P. (1987), "*Domain Names – Implementation and Specification*", Internet Engineering Task Force (IETF) RFC 1035, November 1987.

Monitorware (2009), www.winsyslog.com/en/product/ (Accessed 03 March 2011)

NSTISSAM (1999), "*The Insider Threat To US Government Information Systems*", U.S. National Security Telecommunications And Information Systems Security Committee, NSTISSAM INFOSEC /1-99.

Oracle Corporation (2010), "*System Administration Guide:Security Services*", Solaris 10 Operating System, Part No: 816–4557–19 , September 2010, pp. 559-672,dlc.sun.com/pdf/816-4557/816-4557.pdf, (Accessed 03 March 2011)

Oracle MySQL portal (2010), www.mysql.com (Accessed 03 March 2011)

Pogue C., Altheide C., Haverkos T. (2008), “*Unix and Linux Forensic Analysis DVD Toolkit*”, Syngress, 2008, ISBN: 978-1-59749-269-0.

Probst C., Hunker J., Bishop M., Gollman D. (2009), “*Countering Insider Threats*”, ENISA Quarterly Review Vol. 5, No. 2, June 2009, pp. 13-14.

Psacct utilities (2003), Utilities for process activity monitoring, linux.maruhn.com/sec/psacct.html (Accessed 03 March 2011)

Rivest R. (1992), “*The MD5 Message-Digest algorithm*”, Internet Engineering Task Force (IETF) Request For Comment (RFC) 1321, April 1992.

RLIKE RegExp (2008), “*String Regular Expression Operator*”, MySQL 5.1 Manual, Oracle Corporation,dev.mysql.com/doc/refman/5.1/en/regexp.html (Accessed 03 March /2011)

Trusted BSD Project portal (2009), “*OpenBSM: Open Source Basic Security Module (BSM) Audit Implementation*”,www.trustedbsd.org/openbsm.html (Accessed 03 Match 2011)

Insider Threat Specification as a threat mitigation technique

Introduction:

This chapter describes the Insider Threat Prediction Specification Language (ITPSL), a research effort to address the description of threat factors as a mechanism to mitigate insider threats.

Section A 'The insider threat problem' provides some necessary definitions about the insider misuse problem.

Section B 'ITPSL: Scope, development paradigms and design criteria' starts with a discussion of two intrusion specification language paradigms that influenced the development of ITPSL (subsection B1). The second subsection B2 is concerned with taxonomic and threat modeling research and development efforts designed to address insider threats, with emphasis on abstracting the domain of insider misuse and shaping the threat metrics the language can express. Subsection B3 explains the problems ITPSL is trying to solve and its design criteria. Finally, subsection B4 discusses the programming paradigm that could facilitate the ITPSL construction.

A) The insider threat problem:

The problem of insider IT misuse (the term 'misuse detection' or 'misuse' is also used in the literature) is a serious threat for the health of IT infrastructures. A threat in an IT infrastructure context is "a set of circumstances that has the potential to cause loss or harm" [1]. As a result, in legitimate user context, these circumstances might involve intentional IT misuse activities such as targeted information theft, introducing or accessing inappropriate material, and accidental misuse (e.g. unintentional information leak). In addition, there is also potential for flaws in the design and implementation of the computer system, which could render it susceptible to insider misuse.

Numerous people have tried to define the term "insider" in the context of Information Security. This is because there are many possible sub-contexts that are applicable to shedding light on different aspects of what an insider is and what she can do. For instance, an aspect of insiders relates to what they are allowed and not allowed to do in an organizational context. This is often dictated by the organization's IT usage policy, "a set of laws, rules, practices, norms and fashions that regulate how an organisation manages, protects, and distributes the sensitive information and that regulates how an organisation protects system services" [2] . Insiders that do not follow the rules of the IT policy are formally considered as misusers.

Other definitions focus more on the attributes of an insider, from an organizational trust point of view [3]:"An insider is a person that has been legitimately empowered with the right to access, represent, or decide about one or more assets of the organization's structure". This definition has a wide perspective and emphasizes a key aspect of an insider: that of trust. Trust is a property that goes beyond an IT system oriented view (system credentials, actions, indications). Whilst people who constitute direct threats might not have access to IT access credentials, they still can decide on policies, equipment procurement and other issues that can affect the well being of an IT infrastructure. A good example is an IT director that spends millions on a state of the art security system but does not bother to emphasize or make policies that dictate the flow of information inside the organization (employ that bypasses the system with a simple USB key, intentionally or accidentally).

However, trust has an impact on IT level credentials. A narrower but IT system specific definition can also be useful, in order to focus on insider actions that can be detected by system methods. Hence, an insider is a person that has been legitimately given the capability of accessing one or many components of an IT infrastructure (hardware, software and data) enjoying effortless login by interacting with one or more authentication mechanisms. The word 'legitimately' differentiates the user from an external cracker that masquerades as the user by means of bypassing the authentication mechanisms. The implication of 'effortless' is that an insider does not need to consume time and

effort to gain access to a system resource. This also means that they enjoy trust, a vital property of all insiders.
The reader can consult references [4] to [11] for a detailed qualitative and quantitative review of insider misuse cases.

B) ITPSL: Scope, development paradigms and design criteria
B1) An overview of two intrusion specification languages:

The main function of ITPSL concerns insider threat specification. Threat specification is not a new concept in the information security world. Techniques to describe threats exist for an entire range of information security products, from anti-virus software to several intrusion detection/prevention systems, where specified rules are used to describe a particular range of threats. However, this subsection focuses on generic threat specification. Most products might focus on specific types of threats (anti-virus products relate to malware detection, IDS products might focus on network threats, etc).

When it comes to generic intrusion specification languages, we have two notable examples. The Common Intrusion Specification Language (CISL) [12] and Panoptis [13]. The next paragraphs are going to describe these two languages and discuss their significance for ITPSL.

CISL consists of a semantic framework to unambiguously describe intrusive activities together with proposed data structures that store the event information and can form standardized messages exchanged by various Intrusion Detection System (IDS) components. The semantic representation of intrusive activities is achieved by the formation of an S-Expression. This is a recursive grouping of tags and data, delimited by parentheses. The tags provide semantic clues to the interpretation of the S-Expression and the data might represent system entities or attributes. For this reason, the tags are also called Semantic Identifiers (SIDs).

The best of way of illustrating how CISL works is by considering an example. The statement (Hostname ‘frigg.uio.no’) is a simple S-Expression. It groups two terms, without semantically binding them. One can guess that it refers to a computer system with the FQDN name ‘frigg.uio.no’, but the true meaning of the statement is still vague. In fact, the full semantic meaning of S-Expressions becomes apparent when one forms more complex S-Expressions, by means of combining several SIDs into a sentence.

Figure 1 illustrates a suitably crafted CISL intrusion specification which could be translated in the following plain English translation:

“On the 24th of February 2005, three actions took place in sequence in the host ‘frigg.uio.no’. First, someone logged into the account named 'tom' (real name ‘Tom Attacker’) from a host with FQDN 'outside.firewall.com'. Then, about a half-minute later, this same person deleted the file '/etc/passwd' of the host. Finally, about four-and-a-half minutes later, a user attempted but failed to log in to the account 'ksimpson' at 'frigg.uio.no'. The attempted login was initiated by a user at 'hostb.uib.no'.”

```
(And

        (OpenApplicationSession
            (When
                (Time 14:57:36 24 Feb 2005)
            )
            (Initiator
                (HostName 'outside.firewall.com')
            )
            (Account
                (UserName 'tom')
                (RealName 'Tom Attacker')
                (HostName 'frigg.uio.no')
                (ReferAs 0x12345678)
            )
            (Receiver
                (StandardTCPPort 22)
            )
        )
        (Delete
            (World Unix)
            (When
                (Time 14:58:12 24 Feb 2005)
            )
            (Initiator
                (ReferTo 0x12345678)
            )
            (FileSource
                (HostName 'frigg.uio.no')
                (FullFileName '/etc/passwd')
            )
        )
        (OpenApplicationSession
            (World Unix)
            (Outcome
                (CIDFReturnCode failed)
                (Comment '/etc/passwd missing')
            )
            (When
                (Time 15:02:48 24 Feb 2005)
            )
            (Initiator
                (HostName 'hostb.uib.no')
            )
            (Account
                (UserName 'ksimpson')
                (RealName 'Karen Simpson')
                (HostName 'frigg.uio.no')
            )
            (Receiver
                (StandardTCPPort 22)
            )
        )
    )
```

Figure 1: CISL sentence syntax example

The particular CISL sentence describes a malicious attack that erases an important system file of a UNIX system and consists of three multi-SID S-Expressions. In general, a sentence can be formed by one or more S-Expressions nested at different levels.

Verb SID’s are joined together in a sentence by conjunction SIDs. In the previous example of Figure 1, ‘And’ is the conjunction SID that holds together the three SIDs that form the sentence. In addition, a CISL sentence might employ role, adverb, attribute, referent and atom SID types. Role SIDs indicate what part an entity plays in a sentence (such as ‘Initiator’). Adverb SIDs provide the space and time context of a verb SID. Attribute SIDs indicate special properties or relations amongst the sentence entities, whereas atom SIDs specialise in defining values that are bound to certain event instances (for instance ‘Username’). Lastly, referent SIDs allow the linking of two or more parts of a sentence (‘Refer to’ and ‘Refer as’). There are additional SID types, but the aforementioned ones are the most commonly employed.

One can clearly observe a structural hierarchy for forming complex sentences that also contributes to the semantic meaning. This semantic structure is inspired by the syntax of natural languages. A verb is always at the heart of every sentence and is followed by a sequence of one or more qualifiers that describe the various entities that play parts in the sentence, or qualify the verb itself.

CISL [12] is not only about semantic rules. Its authors were concerned with the encapsulation of the structured semantic information into the ‘Generalised Intrusion Detection Object’ (GIDO), data structures that hold the encoded event information. The purpose of encoding the information in a standard way is to make the process of exchanging the information amongst various IDS components easy.

Unfortunately, despite the well-conceived interoperability target, the CISL GIDO encoding process introduced many problems. Doyle [14] has criticized many of the aspects of the CISL GIDO structure. Although the purpose of the document was to evaluate the fitness of CISL for use in the DARPA Cyber Command and Control (CC2) initiative, the paper identifies serious inadequacies that concern the CISL time resolution data representation facilities, as well as data throughput limitations caused by the fixed size of the GIDO data structure. Finally, Doyle comments on the lack of support for the next generation Internet Protocol (Version 6). Whilst these points are fair, they could easily be corrected by making the necessary changes to the relevant data types and overcome the perceived obstacles. In fact, section 7 of the CISL standard [12] contains specific guidelines that explain how to add information to a GIDO, to clarify or correct its contents. This suggests that the encoding principles are certainly extensible.

A more serious aspect of Doyle’s critique [14] refers to the semantic structure of the CISL language. In particular, his criticism that CISL has “no facilities for representing trends or other complex behavioral patterns; ill-specified, inexpressive, and essentially meaningless facilities for representing decision-theoretic information about probabilities and utilities” indicates that the language would be a bad choice for describing threat prediction related information. The basic reasoning behind this critique is that CISL is too report-orientated and threat mitigation requires a different level of information, not just mere report structures of what is happening on one or more systems. These indeed represent more serious limitations that would require a more radical re-design of the CISL.

In addition to Doyle's criticisms, from a threat specification perspective, we note the following omissions/weaknesses in CISL:

- Inability to express variability in intrusion events: For example, all the necessary time patterns to specify recurring events of significance: The 'When', 'Time' and others SIDs can bind an event to an accurate time and date location. However, this is of little value to a threat specification as the accurate time of an intrusion event is rarely known. An SID operant such as 'afternoon-hours', 'evening-hours' would be more functional. This is true for other type of SIDs such as 'FileSource', network SIDs, etc. The expressions clearly lack the necessary polymorphism required to describe a range of possible events.

- The nesting of S-Expressions does not facilitate logical operands/operators, in order to describe alternative events. This affects the overall polymorphic description at event level.
- General lack of a mechanism to express confidence of a particular metric: Decision theoretic information is a desired feature of threat specification. The process of specifying a threat might include the description of various events. Not all of them have the same level of reliability and as such, a language that omits a mechanism of expressing a confidence in a particular event is a serious issue. This omission also hinters the ability to build up user profiling information.

Nevertheless, Amoroso [15] characterizes CISL [12] and its associative CIDF framework [16] as a "good piece of computer science", despite the fact that it has not managed to infiltrate the IDS/IPS vendor market as a product interoperability platform. CISL is significant for the development of ITPSL for the following reasons:

- It's the first language framework for generic intrusion specification with system interoperability as its design goal, attempting to bridge the gap between language semantics and operating system/IDS product implementation details. This is a desirable feature because a good threat specification mechanism should focus on the threat itself and less on platform specific issues.
- It introduces the S-expression as a way to group the SIDs with the actual data in a hierarchical semantic notation which can nest expressions. Despite the previously discussed weaknesses of its proposed semantic identifiers, the suggested combination increases the clarity of the expression and the S-expression nesting capability increases the specificity of the statement in a consistent manner (the more S-expressions nested together in a the more specific the conditions of the match).

Panoptis [13] is another interesting and more recent intrusion specification language paradigm. The language sits on top of an anomaly detection system which parses standardized UNIX audit process logs. After establishing a user profile based on a number of different criteria, the audit logs are parsed and then checked against the profiling data. A sample of the entities and quantitative criteria that the panoptis system checks against is given below. These include:

- Discrete entities are organized in database tables such as:
    - tty UNIX terminals.
    - uid Users.
    - uidtty Users logged in on a specific terminal.
    - comm Commands.
    - uidcomm Users executing a specific command.
- Process accounting data such as:
    - maxaxsig Signal exit status.
    - maxhog Maximum CPU hog factor (CPU time over elapsed time).
    - maxmem Maximum memory usage.
    - maxavrw Maximum average disk block input/output.
    - maxstime Maximum system time.
    - minbmin Minimum daily start time (start time whithin the 24 hour interval).
    - maxutime Maximum user time.
    - maxbmin Maximum daily start time.

- maxasu Superuser status.
- maxcount Maximum number of times a given record has appeared in the database.
- maxrw Maximum disk block input/output.
- maxacore Core dump flag.
- maxavio Maximum average character input/output.
- maxafork Fork status.
- maxetime Maximum clock time.
- maxavmem Maximum average memory usage.
- maxio Maximum character input/output.

In essence, panoptis is an anomaly detection system envisaged to detect a number of attacks such as data leakage, wiretapping and user masquerading amongst others. The semantics are restricted to configuration file options such as the one illustrated by Figure 2. Declarations of the type *variable=value* and a *keyword(entity, value(s))* combination make up the syntactical convention.

```
     HZ = 100                # "Floating point" value divisor
     bigend = FALSE
     map = TRUE
     EPSILON = 150           # New maxima difference threshold (%)
     report = TRUE             # Set to TRUE to report new/updated
entries
     unlink = FALSE          # Set to TRUE to start fresh
     # Reporting procedure
     output = ’| /usr/bin/tee /dev/console | /bin/mail root’
     # Databases and parameters to check
     dbcheck(tty, minbmin, maxbmin, maxio, maxcount)    # Terminals
     dbcheck(comm, ALL)  # Commands
     dbcheck(uid, ALL)   # Users

     dbcheck(uidtty, maxcount)     # Users on a terminal
     dbcheck(uidcomm, minbmin, maxbmin, maxutime,maxstime, maxmem,
maxrw, maxcount, maxasu)
     # Map users and terminals into groups
     usermap(caduser, john, marry, jill)
     usermap(admin, root, bin, uucp, mail, news)
```

Figure 2: A configuration file sample showing the DSL syntax of 'panoptis'

For instance, the statement *dbcheck(tty, minbmin, maxbmin, maxio, maxcount)* will check the UNIX terminal entity activity against the normal minimum and maximum startup time, as well as the maximum character input/output and the maximum number of times a given record has appeared in the database. If any of these figures exceeds the preset epsilon threshold normal value by 150% (*EPSILON=150* declaration), the observation will be flagged as an intrusion. Note that these checks will be performed against the records of certain users as defined by the *usermap* statements (*caduser, john, marry, jill* as user group 1 and *admin, root, bin, uucp, mail, news* as user group 2).

The simplistic semantics of panoptis suffer from many of the previously discussed drawbacks of CISL. Development on the 'panoptis' system has been discontinued and thus, it is not fair to really judge the effort on the basis of the presented system. The panoptis authors were more interested to present a paradigm whose scope was to parse system audit logs and not a full intrusion specification language.

However, the panoptis approach is an important paradigm for ITPSL for two reasons:

- It is one of the first specification language approaches that target insider misuse incidents. The authors claim that under certain conditions, panoptis could "detect an employee transferring inordinately large amounts of datato a computer outside the organisation even if that employee had proper system authorisations to perform."[13]
- It is one of the first systems that employs a Domain Specific Language approach, in order to design the intrusion specification semantics and capture precisely the domain's semantics. Section B3 will examine the Domain Specific Language more closely.
- It makes use of simple data management techniques in terms of having access to structured data input (reading from UNIX process audit logs) and arranging the anomaly detection threshold in a simple non-relational database. This indicates the need for properly storing and readily accessing intrusion information, an important requirement of a threat specification tool in itself.

For all these reasons, both CISL [12] and Panoptis [13] play an important role to the development of ITPSL.

B2)Insider misuse taxonomies and threat models:

Apart from the process of designing the semantics of the specification language, there are other steps that concern the language designer. In fact, Magklaras and Furnell [17] propose a methodology for deriving a Domain Specific Language for insider misuse detection and prediction that includes three important steps:

- the abstraction of the domain, which involves the removal of all the unnecessary details of the environment;
- the systematic categorisation of the necessary (abstracted) details into language semantics;
- the process of engineering the developed semantics into software.

The domain abstraction is a critical part of the overall language design process and raises the question of which entities and data are relevant to the insider threat domain. The CISL [12] and Panoptis [13] paradigms proposed certain types of data to monitor, without giving a concrete explanation on why these data were chosen and how they can aid the threat detection/prediction process. Section A presented notable insider misuse case studies and surveys and concluded that whilst generic trends can be spotted, this is not enough information to have a concrete picture of the problem. In order to select with confidence a range of insider misuse threat descriptors, a more systemic view of the problem is needed.

Information security taxonomies and threat models provide the answer to these questions. Taxonomies are efforts to classify information. Threat Modeling attempts to make use of the systemic knowledge of the taxonomies and estimate threat levels and/or simulate threat scenarios to help insider misuse researchers understand better important concepts and ultimately estimate threat levels. A model is an "abstraction of the system being studied rather than an alternative representation of that system" [18]. This abstracted representation of the system should closely resemble its real-world behavior. The process of abstracting a real-world situation implies that not all information about its attributes and functions is transferred into the model. Only those attributes and functions that are relevant for the study of certain aspects of the entities involved are included. Thus, insider threat model designers need to consider carefully which attributes and behavioral characteristics of a legitimate user are important to a threat estimation process. Latter paragraphs of this subsection will discuss these issues in more detail.

As with intrusion specification languages (subsection B1), intrusion specification taxonomies are not a new idea in the information security field. An overview of intrusion specification taxonomies is provided by Furnell et al [19]. Amongst these taxonomies, one that specifically addresses insider IT misuse incidents is given by Tuglular [20]. Tuglular's paper is one of the first to suggest a 'target-type of threat' association as a way to prevent insider misuse. The target is an 'asset' and the rule is called a 'strategy' in the terminology he proposes. The suggestion forms the basis for a methodology to predict insider misuse threats. If one can associate successfully certain actions to threats then it establishes the first step towards systematizing insider IT threat prediction.

Most research efforts in the field of intrusion taxonomy classification are still at an early stage. The Tuglular taxonomy, and others mentioned in [19], are useful for the systematic study of intrusions, but they offer little help to a process designed to automatically detect intrusive activities. This is because the classification criteria employed by these taxonomies cannot be qualified or quantified very easily by an Intrusion Detection System with the level of information they exhibit. Moreover, none of these taxonomies is tailored for the process of estimating the likelihood of Insider Threat.

The best way of enhancing the expressiveness of an intrusion taxonomy scheme for insider misuse activities is to focus on the human actions and how their consequences impact the elements of the IT infrastructure that are being targeted. The idea is that it is easier to detect which particular element is affected by a potentially intrusive action, rather than focusing on the task of sensing the motives for initializing an attack or focusing on other non-system detectable factors of the insider misuse domain.

The perplexing variable nature of insider IT misuse is a fact ([4] - [11]). What is considered as misuse by a well-defined IT usage policy is not the same across different organizations. The freedom of the security architect to choose what can be considered as an Insider IT misuse threat indicator and even decide on the confidence of each indicator is important. Most taxonomies enforce a rigid framework for classifying phenomena with clear borders of distinction that offer little space for subjective or varying interpretation of facts. This schema does not fit the case of Insider IT misuse prediction.

Figure 3 below displays the top level of the taxonomy structure indicating the three primary, non-mutually exclusive levels that address these consequences.

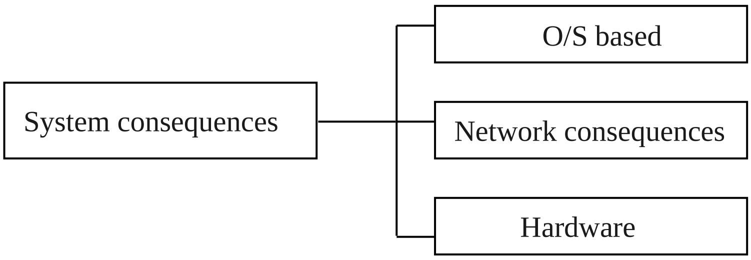


Figure 3: Top hierarchy level of an insider misuse taxonomy

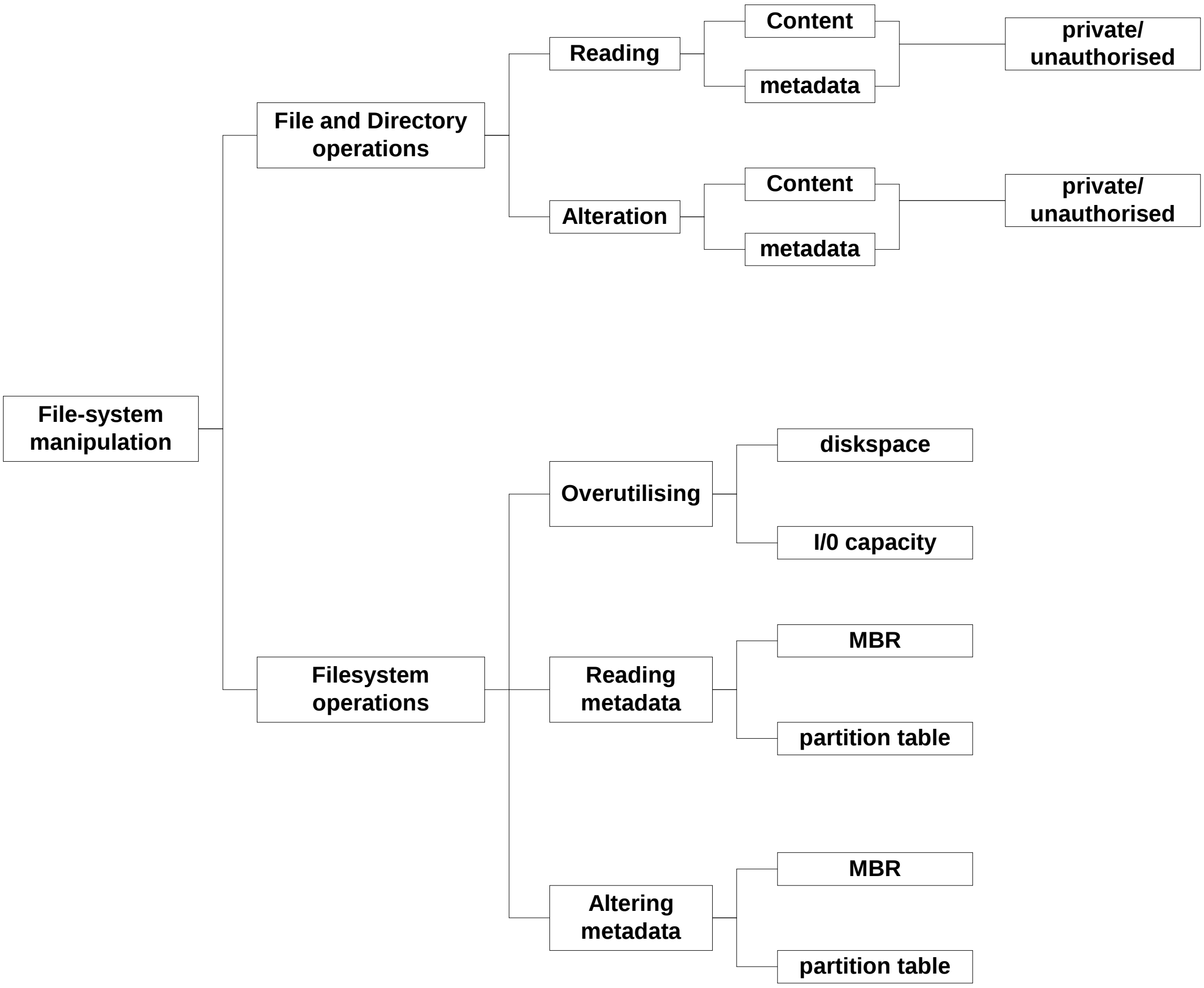


Figure 4: File-system manipulation O/S consequences

The Operating System (O/S) based consequences are branched down to two sublevels of file-system and memory manipulation, illustrated by Figures 4 and 5 respectively. A justification for this is that a large number of security faults [21] involve filesystem and memory management issues, and indeed the core modules of UNIX [22] and Windows-based [23] operating systems provide (amongst others) specific support for the related functions. Hence, it is safe to assume that these two kernel functional attributes can be used as a strong criterion for further classifying legitimate user activities.

At File/Directory level, a misuser may attempt to read or alter (write/create) certain files. These files might contain sensitive or unauthorised information (information theft or fraudulent modification of vital information). A knowledgeable insider might also attempt to read or modify file information that is not directly related to its content. Bach [22] and Richter [23] emphasize that most Operating Systems allow a file to contain additional information such as access/creation/modification times as well as information that relates the file to its owner and permits access to it under certain conditions. Although the mechanisms that implement these file attributes are different amongst Operating Systems, they are collectively known as file metadata and they are vital mechanisms to secure the privacy, availability and integrity of the file contents. Consequently, they are good candidates for exploitation by a legitimate user who is about to perform a deliberate or accidental misuse act.

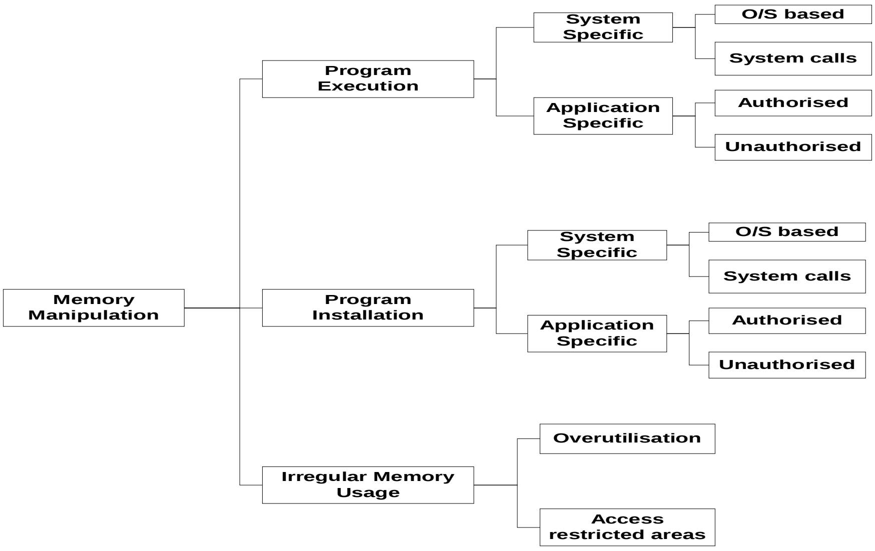


Figure 5: Memory Manipulation O/S Consequences

The points mentioned in the previous paragraph are also valid for 'filesystem' related data. Every Operating System organizes its files and directories by means of a specific set of rules that define how a file (contents and metadata) are about to be stored on the physical medium. The Operating System sub-modules that handle these issues are known as filesystems. Attempts to read or alter the physical medium's Master Boot Record (MBR), intentional or accidental modification of partition table data are some of the most notable auditable actions that could point to legitimate user misuse acts. Robert Hanssen's attempt to hide information in modified floppy disks, a case [4] discussed in section A, is a classic reminder of this kind of activity.

In addition to filesystem content and metadata modification, a survey of insider misuse conducted by Magklaras and Furnell [11] showed that excessive disk space consumption is perceived as a problem for many of the respondents. Under certain conditions that depend on the configuration of the IT infrastructure, a legitimate user might produce a deliberate or accidental Denial of Service attack (DoS).

Memory inspection is the best way to see if a legitimate user attempts to run or even install a suspicious program. Indeed, it is one of the core techniques used in the detection of overtly malicious code, such as viruses and Trojan horse programs. The usage of unauthorised programs is a serious issue that can also create a way for accidental misuse by introducing a number of system vulnerabilities, as described by Papadaki et al [24]. The execution or installation of these programs could be intercepted by either recognising a program's footprint in memory or by intercepting a well-known series of system calls produced by various suspicious programs. For example, the fact that a non-advanced user is trying to compile an advanced vulnerability scanning tool is an event that should be noticed and serve as a good indicator of potential misuse activities that are about to follow.

In addition, attempts to consume large memory portions of an operational system that are related to a legitimate user account can serve as good indicators of (intentional or accidental) insider misuse at Operating System level. One might argue that the 'irregular memory usage' sub-categories should really belong under the 'Program execution' hierarchy of events. However, it is possible that someone will produce a quick and easy Denial of Service attack on a running system by forcing the host to commit large portions of system memory to a process, as demonstrated in various case studies described by Moore et al [25]. Moreover, a large category of security faults can be achieved by means of accessing normally restricted memory areas, creating what is commonly known as a “buffer overflow” attack [26]. As a result of these issues, it was felt that a separate sub-category hierarchy should exist to describe these events (Figure 5).

The filesystem and memory manipulation consequences conclude the O/S consequence category of the proposed taxonomy (Figure 3). The next category, “network consequences”, represents another distinct set of factors that could be taken into consideration in order to classify insider misuse threat indicators. Figure 6 illustrates the network-related consequences of acts that could be used as legitimate user threat indicators.

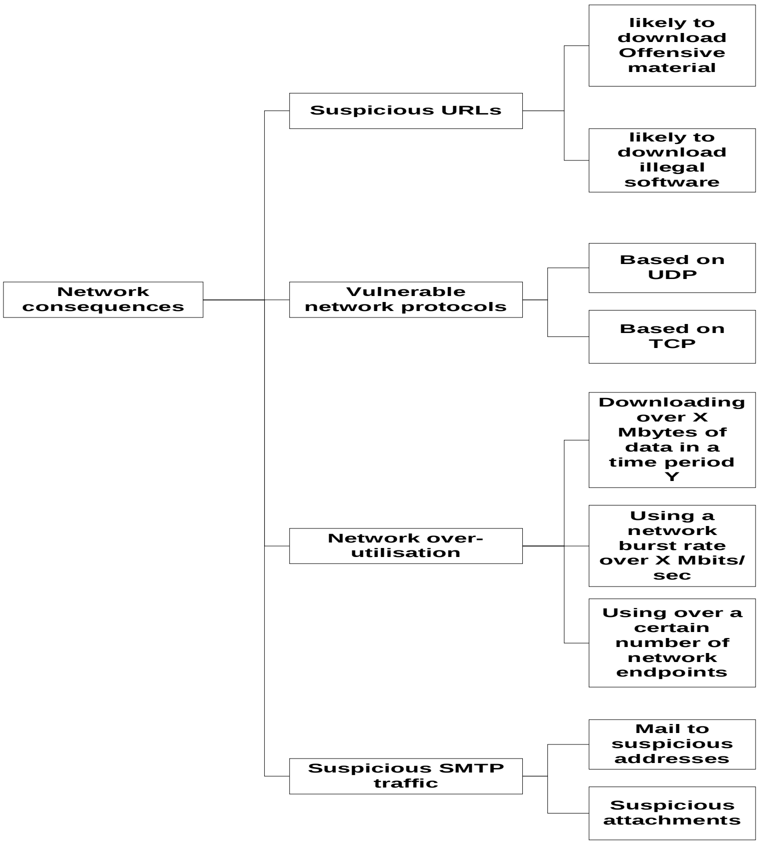

Network consequences
Suspicious URLs
likely to download Offensive material
likely to download illegal software
Vulnerable network protocols
Based on UDP
Based on TCP
Network over-utilisation
Downloading over X Mbytes of data in a time period Y
Using a network burst rate over X Mbits/sec
Using over a certain number of network endpoints
Suspicious SMTP traffic
Mail to suspicious addresses
Suspicious attachments

Figure 6: Network consequences of the insider IT misuse prediction taxonomy

Network packets that are associated with certain legitimate users and indicate the usage of a variety of network protocols and applications that might introduce certain vulnerabilities are also distinct ways of accidental or intentional IT misuse. For example, it could be said that a user that employs the TELNET [27] protocol to login to a multi-user system is more likely to have her account compromised than a user who logins via the Secure Shell (SSH) application [28] due to the fact that the earlier application transmits the user password in clear-text form across the network, whereas the latter one encrypts it.

Someone might also like to differentiate between TCP and UDP based applications/protocols. From a potential threat point of view, UDP services are less secure than TCP based ones. For example, Ziegler [29] discusses in detail how UDP’s lack of flow control and state mechanisms can create various data security problems. Consequently, the distinction between the usage of UDP and TCP services can serve as a potential insider misuse threat indicator, on the basis that UDP services are more likely to be accidentally (or intentionally) abused by a legitimate user.

Although the 'Filesystem Manipulation' subcategory of the taxonomy indicates ways with which disk storage capacity can be misused, the results of over-utilisation can also affect network capacity. For instance, a legitimate user could start downloading massive quantities of data, exceeding the network bandwidth cost budget of a business (Downloading over X Mbytes of data in a period Y). The X and Y number limits can be selected by the network administrator according to the company budget requirements.

In addition, a legitimate user might also cause network congestion by exceeding the data network's 'burst' or throughput capacity or exhausting the number of available network endpoints, as described by Sharda [30]. Bandwidth hungry applications, such as video streaming players, and multiple data transfers can cause congestion that can severely impact the performance of a data network or affect the Quality of Service (QoS) of certain applications that require sustained data network throughput.

Finally, incoming or outgoing SMTP headers or attachments might indicate activity related to e-mail misuse that can certainly be traced in network or host level. Outgoing e-mails that contain a set of particular files as attachments (e.g. password database files, other sensitive material) and have unusual destination addresses (e.g. unknown Hotmail accounts, a large number of recipients) should serve not necessarily as intrusion indicators but as insider threat estimators. The plethora of malicious code efforts and phishing techniques may have an external origin, but the threat is realized by the actions of unsuspecting legitimate users. In addition, proprietary information theft could also be realized by means of emailing sensitive material to non-authorized external entities.

The last system consequences category ("hardware") plays an important role in preventing a number of computer system threats. Insiders can often access the physical hardware of the machine very easily. Thus, removal or addition of hardware components, as well as modifications of their default configuration are some of the events that may act as important indicators of insider misuse prediction in a computer system.

However, in order to make use of these threat indicators, we need a way to quantify them. This paves the way for the discussion of various Insider Threat Models presented in the following paragraphs.

The first important step of deriving an Insider Threat Prediction Model is to decide which attributes and behavioural (functional) characteristics of a legitimate user are important to the Threat Estimation Process. This will produce a set of Insider Threat Qualification Attributes (ITQAs).

The next step in the process of establishing the model is to describe how the ITQAs can be quantified, in order to estimate the level of insider threat per individual user. This will involve the establishment of a suitable mathematical function, which will take as input a number of ITQAs and will associate them with a certain level of threat. We shall call this function the Estimated Potential Threat function, which quantifies the ITQAs. At this point, the overall target of our ideal model will be achieved: the establishment of a mechanism that will map ITQAs to certain threat levels.

The development of insider threat models is a relatively new idea. Wood [31] provides an excellent basis for qualifying a set of metrics to mitigate insider threat. Most of these criteria are in line with the conclusions issues discussed as part of the insider misuse taxonomy discussed in the previous section.

In particular, Wood suggests that a malicious insider can be qualified in terms of distinct attributes:

- Access: The insider has unlimited access to some part or all parts of the IT infrastructure and the ability to physically access the equipment hardware. Consequently, the insider can initiate an attack without triggering traditional system security defences.

- Knowledge: The legitimate user is familiar with some or all the internal workings of the target systems or has the ability to obtain that knowledge without arousing suspicion.
- Privileges: The malicious insider should not have problems obtaining the privileges required to mount an infrastructure attack.
- Skills: The knowledgeable insider will always have the skills to mount an attack that is usually limited to systems that he/she is very familiar with. The model assumes that a given adversary is unlikely to attack unfamiliar targets.
- Tactics: This attribute refers to the methods used to launch the malicious attack. They are dependent on the goal of the attack and might include a variety of scenarios such as plant-hit-and-run, attack-and-eventually run, attack-until-caught as well as passive information extraction acts.
- Motivation: Insiders might launch the attack for profit or sabotaging the target organisation. Some of them might mount an attack for personal reasons such as taking revenge against the enterprise or even satisfy their plans to invoke some policy change inside an organisation.
- Process: The model assumes that a legitimate user follows a basic predictable process to mount an attack that consists of distinct stages. First the malicious adversary will become motivated to mount the attack. The next logical stages involve the identification of the target, the planning of the attack and finally the act of mounting the attack itself.

All of the previously mentioned attributes emphasize important aspects of the insider misuse problem. However, Wood's criteria do not necessarily represent a clear picture for the establishment of an insider threat prediction model. Not all stages of an insider attack can be safely predicted. Some of the previously mentioned attributes are difficult to qualify by an Intrusion Detection System. The 'motivation' adversary attribute is one of them.

It is very difficult to establish a set of sensors that could reliably deduce when an individual becomes motivated to misuse a system. For instance, let us suppose that IDS sensors record that a commercially important file is transferred from a disk to an external storage medium in the early morning hours. The fact that this particular file transfer took place could be related to a malicious act or an innocent file backup process performed by the system administrator as part of a system recovery process. It is important to maintain a record of these types of events, but their existence does not necessarily indicate an insider misuse event in progress. The plethora of the potential origins of such an event would increase the amount of information to be evaluated. Consequently, the complexity of the algorithms to capture and evaluate this type of information would deem this attribute's exploitation impractical.

If someone observes the different stages of the 'process' insider-modelling attribute, it becomes clear that the closer we get to the actual attack itself, the stronger the indicators of insider threat. Although detecting motivation might be tricky, with a carefully chosen quantification scheme of ITQAs, someone could sense an adversary during the target identification and attack planning stages. This strategy goes along the line of thinking of our proposed misused taxonomy being based on system-level factors.

In addition, other attributes seem to be so closely related that might be redundant. For instance, it would be more logical to combine the attributes of 'access' and 'privileges' into one 'insider access rights'. The issue of obtaining a privilege to mount an attack should include logical and physical means of interacting with the systems. The same could be said for the attributes of 'knowledge' and 'skills', because the ways in which a legitimate user gets to know a system and what can be inferred from the insider's system knowledge are issues that are closely interrelated.

A more recent research effort by Schultz [32] presents a preliminary framework for understanding and predicting insider attacks by providing a combination of behavioural and system usage ITQA metrics. The paper mentions the detection of system usage patterns that may act as "signatures" of a legitimate user or certain indicators of an attack preparation (“deliberate markers” and “preparatory behaviour”). Legitimate users might also make noticeable mistakes in the process of misusing a system (meaningful errors). Finally, “correlated usage patterns” refers to sequences of actions that might not be detected in individual systems but they could certainly indicate misuse when considered against multiple systems.

Schultz also suggests that certain aspects of a legitimate user’s personality could serve as threat indicators. In particular, on-line (e-mail, IRC or other forms of computerised human-to-human communication) verbal behaviour with signs of aggression, dominance towards particular people might serve as a good prognosis factor of certain attacks (“verbal behaviour”). Furthermore, based on the works of Shaw et al [33], the research suggests that it is possible to examine other “personality traits” as potential threat indicators.

The Schultz preliminary framework even suggests a way to quantify all these metrics by means of a multiple regression equation that consists of the summation of the ITQA metric variables multiplied by their weightings. If X1, X2, X3… XN represent the quantified ITQA metrics, Wi (i=1, i=N) their respective weights and C an arithmetic offset constant, then the expected estimated threat Xe is derived in figure 7 below:

$$\mathbf{Xe = (\Sigma\ WiXi) + C = W1.X1 + W2X2 + W3X3 + \ldots + WNXN + C}$$

Figure 7:Schultz threat model equation

One notable absence of the Schultz insider threat prediction scheme is that there is no direct association between the estimated level of threat and the legitimate user’s level of technical knowledge. Although the proposed metrics can provide evidence that could be used to infer the level of user sophistication, there is no mentioning of a mechanism that takes that into consideration. Given the fact that, at the time of writing, the field of Insider Threat modelling is premature to reveal any usable results, it is difficult to prove the real impact of user sophistication on the threat level. On the other hand, Wood’s model, a number of case studies and the survey results (section A) provide strong indications that there is a direct relationship between these two concepts. In that sense, the lack of a legitimate user sophistication gauging component could present a serious omission of the Schultz framework.

In addition, the exploitation of future mechanisms that will associate personality traits to potential misuse threat levels raises certain ethical and feasibility concerns. It is outside the scope of the thesis to examine ethical issues and the various laws that are associated with them. Nevertheless, the process of designing a model that is going to be employed in the real world should take into consideration its troublesome aspects. A metric that penalises real people in terms of their character traits will be considered unethical by many and depending on regional legislation may be also not feasible to implement.

In summary, the Schultz framework is more refined than Wood’s earlier Insider Threat model in that it provides more concrete examples of ITQA metrics as well as a basic quantification mechanism for them. However the framework is still in its infancy. The author acknowledges that the chosen metrics need further refinement in order to prove their usefulness in a threat estimation process.

Brancik's [34] seminal work on the insider threat modelling should be referenced as a good source of information. Brancik's efforts center around information alteration, which is an important element of insider fraud, despite the fact that insider misuse surveys indicate that the frequency of these incidents are lower than other most common misuse incidents (web and email abuse). His Tailored Risk Integrated Process (TRIP) is the most important contribution. However, a risk management process deviates from traditional modelling approaches. This is because it focuses on factor evaluation. Detection of threat metrics is not addressed extensively.

Finally, all of the aforementioned research efforts do not address the issue of managing the representation of the data that feed the model component functions. One could argue that a preliminary model design needs to focus more on the scope, quality and quantity of its insider threat modelling functions. On the other hand, a well-thought definition of the procedures that represent and store the data that feed the threat modelling functions may have a notable impact on the computational efficiency and acceptance of the model. The reasons that support the need for this requirement are going to become apparent in the following paragraphs.

For all these reasons, we need a more formalised and broader model description. an Insider Threat Prediction Model that attempts to overcome the shortcomings of previous research work has been published by Magklaras and Furnell [35].

Considering a legitimate user population that has access to various components of an IT infrastructure, the core of the Insider Threat Prediction Model is a three-level hierarchy of mathematical functions evaluated in a bottom-up approach. At the top level, the Evaluated Potential Threat (EPT) function provides an integer value that quantifies and classifies the potential threat for each legitimate user into three different categories. If x denotes the computed EPT for a legitimate user, EPT_MAX a threshold EPT value for considering the user a threat and EPT_MIN a threshold EPT value for considering the user's on line presence as suspicious, then:

- Important internal threat ($x \geq$ EPT_MAX): It indicates a high potential of a particular user misusing the system.
- Suspicious (EPT_MIN $\leq x <$ EPT_MAX): This flags a condition where a particular user behaves in a manner that does not constitute a substantial threat but it is still a concern.
- Harmless ($0 \leq x <$ EPT_MIN): To indicate that the potential of misuse is nearly non existent for a particular user.

$$\mathbf{EPT} = \sum \mathbf{F}_{\mathbf{ITPQA}} \Rightarrow$$
$$\mathbf{EPT} = \mathbf{F}_{\mathbf{attributes}} + \mathbf{F}_{\mathbf{behavior}} \Rightarrow$$
$$\mathbf{EPT} = \mathbf{C}_{\mathbf{role}} + \mathbf{F}_{\mathbf{accessrights}} + \mathbf{F}_{\mathbf{behavior}} \Rightarrow$$

$$\mathbf{EPT} = \mathbf{C}_{\mathbf{role}} + \mathbf{C}_{\mathbf{sysadm}} + \mathbf{C}_{\mathbf{criticalfiles}} + \mathbf{C}_{\mathbf{utilities}} + \mathbf{C}_{\mathbf{physicalaccess}} + \mathbf{F}_{\mathbf{sophistication}} + \mathbf{F}_{\mathbf{fileops}} + \mathbf{F}_{\mathbf{netops}} + \mathbf{F}_{\mathbf{execops}}$$

Figure 8: The Magklaras and Furnell model equation

Each of the threat component functions models particular aspects of insider attributes and behavior. At the moment, in order to devise a well structured organization of threat components, the suggestion is to provide two threat component functions. The first one considers legitimate user attributes such as access rights and professional role, whereas the second evaluates potential threat simply by examining aspects of user behavior at the system level, as shown in Figure 8.

Table 1 lists the maximum weights of the nine top-level EPT formula components that are explained in detail in latter sections of this chapter. Some of these components are constants (Crole, Csysadm…etc) that belong to the $F_{attributes}$ function, whereas others constitute sub-functions of the $F_{behavior}$ function that address the assessment of the legitimate user on-line behavior.

It should be emphasized that the proposed maximum weights on table 1 are not meant to be fixed. A system administrator/security specialist can re-define the maximum weights, in order to reward a particular metric that he trusts more than the others. For this reason, the nine weights of Table 1 constitute the *Weight Matrix,* a very important concept of the ITPM system. The Weight Matrix allows a specialist to further tune the sensitivity of the model, depending on the way he constructs misuse signatures, his confidence on the various metrics and the nature of the incident he is trying to predict. This feature enhances the adaptability of the proposed model scheme and enables to represent decision theoretic information.

EPT Component	**Maximum Weight**	**Meaning**
Crole	6	What is the documented role of the user inside the organization?
Csysadm	6	Has the user access to Operating System administration utilities?
Ccriticalfiles	6	Is it meant for the user to access commercially sensitive files?
Cutilities	6	Can the user execute application critical utilities?
Cphysicalaccess	6	Has the user physical access to critical parts of the IT infrastructure?
Fsophistication	10	How capable is the user in terms of his computer system knowledge?
Ffileops	20	What are the signs of forthcoming insider misuse at file-level?
Fnetops	20	What are the signs of

		forthcoming insider misuse at data network level?
Fexecops	20	What are the signs of forthcoming insider misuse at program execution level?

Table 1: A sample Weight Matrix in the Magklaras and Furnell model

The reader can refer to the Magklaras Mphil thesis [36] for more details of the model and the reasoning behind the design of the proposed threat estimation functions. Two important things from this model should be emphasize here:

- the inclusion of various ITQAs at various levels (file, network, process execution) to represent a variety of system detectable user events.
- The introduced Weight Matrix concept as a mechanism of expressing different levels of confidence for the various ITQAs for a particular threat description.

Both of these things play a great role in the design of the ITPSL. The next section will explore the ITPSL relationship to the threat model process, as well as the overall scope of its inception.

B3) The scope of an Insider Threat Prediction Specification Language (ITPSL):

Information security surveys and notable insider misuse cases reported by mass media were discussed in section A of this report. The earlier paragraphs of section B introduced a more systematic presentation of the insider misuse domain by presenting a suitable insider taxonomy and a resulting insider threat model. However, how a threat model fits with a threat description language is not very clear. Figure 9 illustrates the relationship of the ITPSL and the proposed ITPM model.

The flow of information starts with a security analyst writing a description of the particular insider misuse scenario, using the ITPSL semantics. The signature is validated by a compiler that translates the signature directives to query commands and makes use of an event logging infrastructure, in order to examine whether the ITQAs the signature mentions exist in the system. Apart from the semantics that qualify/quantify the ITQAs, the signature embodies a Weight Matrix statement which indicates the confidence of each specified ITQA. The results are passed to the ITPM engine which then derived an EPT value, indicating that a likelihood of a particular threat.

Figure 9 also includes the security analyst/system specialist both at the beginning of the information flow (signature construction) and at the final stage, where the final assessment is done. This emphasizes that the analyst is in charge of the process, both in terms of defining what constitutes a threat and also in terms of judging whether the likelihood expressed by the model is accurate.

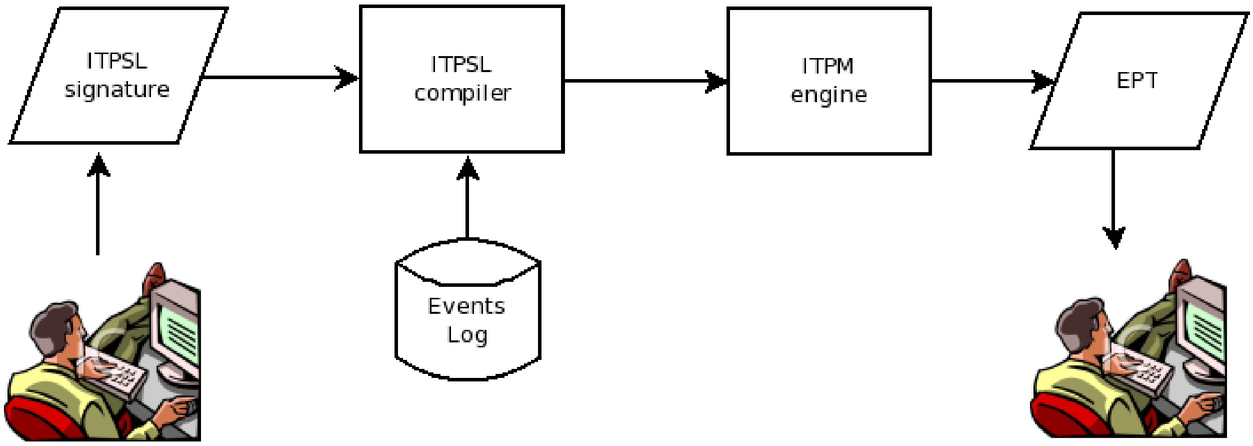


Figure 9: The relationship between ITPSL and ITPM

This places the foundation of the context of ITPSL as a component of an entire Insider Threat management architecture [36]. The model estimates a threat which is described by a language and the likelihood is judged by the IT specialist. The emphasis is on the description and thus, the language addresses the lack of case repositories that express details of insider misuse incidents is apparent. An early report outlining aspects of the insider threat to the US government information systems published by the NSITSSAM Committee [37] considers the absence of case repositories as one of the limiting factors in the field of insider IT misuse mitigation research. In addition, the Carnegie Mellon University CyLab's 'Common Sense Guide to Prevention and Detection of Insider Threats' publication [38] states clearly the need to keep detailed records of employee actions in relation to file access, application usage and network connection matters.

ITPSL could also be a tool for digital forensic investigators. Digital forensics is an important research discipline of the information security field that is concerned with providing evidence to legal proceedings by means of gathering data to determine exact details of various types (internal and external origin) of system attacks. Brancik [34] mentions the importance of suitable tools to produce Key Fraud Signatures (KFS) to aid insider threat mitigation and thus signifies the overlap between insider misuse and the field of digital forensics.

The most widely used form of data forensic investigation is quiescent or static analysis. For such type of analysis, an investigator would utilize a number of toolkits to make a forensically valid copy of the affected system's non-volatile data storage media and perform a "post-mortem" examination of the copied media. The goal is to examine static data (documents, images, email and system files) for digital evidence. AccessData's Forensic Toolkit [39] and Guidance Software's Encase [40] are two well known toolkits that perform, amongst other things, static digital forensic analysis.

However, static digital forensic analysis reveals an incomplete picture of the system in question. It cannot portray accurately the non-quiescent (dynamic) state of the system under investigation. Information such as active network endpoints, running processes, encryption keys for decrypted on-disk content, user interaction data (number of open applications per user, exact commands), as well as the content of memory resident processes may not be recorded accurately on non-volatile media. Hay et al. [41] discusses the shortcomings of static digital forensics analysis in detail. In order to overcome the barriers of static analysis, Adelstein [42] discusses the virtues of non-quiescent or live analysis, which essentially gathers data while the system under-investigation is operational. Microsoft's Computer Online Forensic Evidence Extractor (COFEE) [43] and FATkit [44] are two examples of tools that are able to extract live forensic data from volatile storage locations of a computer system.

Live data forensics analysis fills the gap of static examination methods, but it has its own disadvantages. Carrier [45] and Hay et al [41] discuss the risks associated with acquiring live digital forensic data. In particular, live analysis methods suffer from three basic problems:

- Investigator privileges: The investigator needs administration or escalated privileges to run the live analysis utilities. This could present a number of problems in environments where access policies prohibit escalated privileges from external entities to computer systems.
- System and data integrity:The data gathered during the live analysis phase might be compromised due to system (due to rootkit infection, misconfiguration or intentional alteration of data by one or more system users). Whilst the memory data retrieval issue has been addressed by some complex hardware configurations whose purpose is to reliably acquire volatile data from system memory, the rest of the data acquisition issues are serious and they stem from the fact that data are logged on the system under investigation and not on a safer area before they are analyzed.
- The “observer effect”: When static analysis methods are used, the investigator can examine the data without affecting the source media state. Unfortunately, that is not true for live analysis where the investigator's actions can affect the data. One would have to separate carefully the implications of the investigator's actions from the original data.

The previously mentioned needs shape the scope of the Insider Threat Prediction Specification Language (ITPSL): A specialized language that is able to encode system level data that concern legitimate user actions, in order to aid the process of misuse threat prediction and assist computer forensic officers in the process of examining insider misuse incidents. As such, ITPSL’s target audience is the security analyst/expert, as well as the seasoned IT administrator in charge of system operation and security issues. Both of these types of domain experts should be able to express insider misuse scenarios by using the language semantics to construct signatures of threat scenarios. More specifically, the ITPSL language should be able to meet the following high level functional requirements.:

- FR1:Separate the analysis data from the target system(s) to minimize issues with maliciously or accidentally altering the data.
- FR2:The architecture of the language should facilitate the creation of suitable insider threat signature repositories, so that security specialists/system administration could easily browse for signatures of various threat scenarios. This feature aims to address the lack of suitable case repositories discussed in the earlier paragraphs of this section.
- FR3:Its semantics and logging mechanisms should facilitate the description of both static and live forensic insider misuse system data at the network, process and filesystem layer, in response to the issues discussed .
- FR4:The semantic description of user actions should encompass temporal indicators so that sequences of events could clearly be expressed and logged.
- FR5:The language should be able to represent decision theoretic information to address the criticisms of earlier intrusion specification examples such as CISL [12]. This implies the ability to consistently express various potentials scenarios of insider actions, giving the signature polymorphic properties.
- FR6:The semantics of the language should offer a consistent hierarchical way of describing a variety of scenarios and should be easily readable by humans and software modules.
- FR7: Finally, ITPSL should have an operating system agnostic scope. The signature author should use the same semantics to express the various misuse threat scenarios regardless of

whether the underlying operating system is Microsoft Windows, Linux/Unix, MACOSX or other applicable platform. The language semantics should bridge any gaps created by operating system esoteric peculiarities that could affect the process of expressing threat indicators.

B4) The Domain Specific Languages (DSL) programming paradigm:

The ITPSL scope defines clearly a specific task of expressing insider threat metrics. This paves the way for the selection of a mechanism that allows the language designer to focus on the problem in question. A Domain Specific Language (DSL) is a semantic mechanism tailored specifically for describing the details of a particular task. The main goal is the usage of appropriate semantics to reduce the effort required to reference and manipulate elements of that particular domain.

Spinellis [46] defines a Domain Specific Language as “programming language tailored specifically to an application domain: rather than being for a general purpose, it captures precisely the domain's semantics”. DSL schemata have been employed successfully in a number of different areas. Consel [47] discusses the range of applications that have employed a DSL which includes device driver construction, active networking and operating system process scheduling. Moreover, Eric Raymond [48] outlines some widely known ‘mini’ languages employed in the Unix community (regular expressions, awk, m4) and beyond (Postscript, SNG, Glade) as examples of domain specific languages. This list is by no means exhaustive, as many more DSLs exist today. A DSL is really a framework that offers the ability of building specific and concise notations to express a problem domain, as well as safe (as predictable) code due to semantic restrictions. Both of these properties are very desirable in the process of developing insider threat specifications.

DSLs are also categorized as external and internal in terms of the way they are implemented [49]. External DSLs are discrete systems, independent from any host language and they contain their own interpreter or compiler to parse the language statement and perform post interpretation/compilation actions. In contrast, internal DSLs are semantics embedded inside a general purpose programming language and thus are dependent from the interpreter/compiler of the host language. Examples of external DSLs are the ‘mini’ Unix languages mentioned by Raymond in [48], whereas internal DSL languages tend to be embedded in programming anguages such as Lisp [50], Smalltalk [51] and Ruby [52].

The process of deciding which DSL approach to follow for implementing ITPSL is important. External DSL approaches offer a greater freedom to experiment with the process of constructing insider threat semantics but they provide a higher overhead when it comes to development issues combined with a higher learning curve for the language users. On the other hand, internal DSLs offer less development overhead as parsing, interpretation and compilation issues are handled by the host language environment. If one takes into account that the host general programming language will have already mature semantics and an established user base, it is easy to conclude that an internal DSL would have less steep learning curve than an external DSL approach.

However, the internal DSL dependency on the host language environment might create problems for the language designer. The most important issue might arise from a mismatch between the symbolic integration of the embedded DSL and the general vocabulary of the general purpose host language. General purpose language vocabularies are rich enough to express a variety of scenarios in an abstract way. For example, on a network access scenario, a general purpose programming language vocabulary can express details of the origin and destination of a network connection but not express network access patterns. In that case, if one tries to engineer the additional functionality into the general language, the process of constructing meaningful semantics might be impaired due to the general language syntax or due to the host language underlying data structures that might not be able to represent fully the required domain information.

A secondary practical problem of adopting an internal DSL approach might include parameter evaluation and performance issues. An insider threat prediction operational environment requires the evaluation of various parameters at runtime. If a statically compiled host general language is used (such as C/C++), runtime evaluation of parameters might pose a challenge. There are of course scripting languages [53] where runtime evaluation is not an option, but they might be slow. Ways to combine compile and runtime languages do exist (i.e. a Perl Script calling a C/C++ library via API wrappers), however the complexity of combining domain specific semantics with more than one language should not be underestimated.

For all these reasons, ITPSL follows the external DSL approach allowing for freedom to create the semantics from scratch with commonly changed parameters to be altered without recompilation issues and no dependence on host language idiosyncrasies. The issue of the learning curve for a domain expert to learn yet another language is of course considerable. However, the narrow scope of a DSL language combined with carefully crafted semantics should create a low complexity interface of relatively few (when compared to a general purpose language) statements and thus make the language easy to learn. This approach has been followed by a number of security related research DSLs such as CISL [12] and Panoptis [13], as discussed in previous sections. For now, it should be noted that both of them can be categorized as external DSLs using configuration files to encode statements that have no resemblance to general purpose programming languages.

There are also a number of external DSLs that utilize XML [54] to convey information. Using XML as a markup to construct DSLs is a common approach and thus XML-enabled DSLs are the subject of the next section.

Conlusion

The ability to specify insider threat scenarios can be a useful threat mitigation technique. Starting with suitably crafted insider misuse taxonomies, we develop a standardized vocabulary to describe system-level aspects of insider threat scenarios. This qualifies suitable Insider Threat indicators. Insider Threat models help to quantify the threats and provide a measure of the likelihood of the occurrence of a particular misuse scenario. Finally, a Domain Specific Language Insider Threat Prediction Language (ITPSL) designed to describe these scenarios will be the focal point of a threat mitigation technique. Such a language could also complement forensic tools, acting as a repository of events capable of replaying certain threat scenarios.

The design and construction of ITPSL is a work in progress.

References:

[1] Pfleeger C., Pfleeger S. (2003). “Security in Computing”, 3rd edition, Prentice Hall, Englewood Cliffs, NJ.: Page 6 contains the definition of the term “threat” in an information security context, pages 198-200 and 451 explain the concept of the Access Control List (ACL).

[2] Caelli, W., Longley, D. and Shain, M. (1991), Information Security Handbook, Stockton Press.

[3] Probst C., Hunker J., Bishop M., Gollman D. (2009), “Countering Insider Threats”, ENISA Quarterly Review Vol. 5, No. 2, June 2009, pages 13-14.

[4] Computer Security Institute (2001), “2001 CSI/FBI Computer Crime and Security Survey”, Computer Security Issues & Trends, Vol. VII, NO.1. In addition, the following readers should refer to the following technical discussion on the Slashdot web portal which reveals details of Hansen's data hiding techniques:Slashdot (2001), “Spying and Technology: Robert Philip Hanssen”, posting on Slashdot.org, 22 February 2001, URL: http://slashdot.org/articles/01/02/22/0622249.shtml

[5] Computerworld Internet Portal (2000), “The Cyber-Mod Squad Sets Out After Crackers”, Article written by Deborah Radcliff documenting the case of Abdelkader Smires
URL:http://www.computerworld.com/news/2000/story/0,11280,45927,00.html

[6] Garfinkel S, Spafford G. (1996), ‘Practical UNIX and Internet Security’, Second Edition, O’Reilly and Associates, Sebastopol, CA, ISBN: 1-56592-148-1

[7] ZDnet Internet Portal (2001), “Firms shop around for Net law jurisdictions”, article written by Wendy McAuliffe
URL: http://news.zdnet.co.uk/business/0,39020645,2085983,00.htm

[8] Gray P. (2002), “Australian Universities under fire from hack attacks”, Zdnet portal, article commenting on IT security procedural deficiencies in the academic sensor commens briefly on the University of Oslo hacking incident.
URL: http://www.zdnet.com.au/news/security/soa/Australian-Universities-under-fire-from-hack-attacks/0,130061744,120270076,00.htm

[9] Richardson R. (2008), “2008 CSI Computer Crime & Security Survey The latest results from the longest-running project of its kind”
URL:http://www.gocsi.com/forms/csi_survey.jhtml;jsessionid=PB1OETTDYEQFBQE1GHRSKHWATMY32JVN

[10] (2008) “INFORMATION SECURITY BREACHES SURVEY 2008 | technical report”, published by the Department of Business Enterprise and Regulatory Reform (BERR),
URL: http://www.berr.gov.uk/files/file45714.pdf

[11] Magklaras G., Furnell S. (2004). “The insider misuse threat survey: investigating IT misuse from legitimate users”m Proceedings of the 5th Australian Information Warfare & Security Conference, Perth Western Australia, 25-26 November 2004, pp. 42-51.

[12] Feiertag R., Kahn C., Porras P., Schnackenberg D., Staniford-Chen S., Tung B. (1999), “A Common Intrusion Specification Language (CISL)”, June 1999 revision, URL: http://gost.isi.edu/cidf/drafts/language.txt

[13] Spinellis D., Gritzalis D. (2002), “Panoptis: Intrusion detection using a domain-specific language.”, Journal of Computer Security, 10:159–176, 2002.

[14] Doyle J. (1999), “Some representational limitations of the Common Intrusion Specification Language”, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1999 Revision.

[15] Amoroso, E. (1999), “Intrusion Detection: An introduction to Internet surveillance, correlation, traps, trace-back, and response”, Second Edition, Intrusion.Net books, NJ, ISBN:0-9666700-7-8, June 1999.

[16] The Common Intrusion Detection Framework (CIDF), URL: http://gost.isi.edu/cidf/

[17] Magklaras G., Furnell S., Brooke P. (2006), “Towards an Insider Threat Prediction Specification Language”, Information Management & Computer Security, vol. 14, no. 4, pp361-381.

[18] Somerville I.(2000), “Software Engineering”, Addison Wesley, ISBN: 020139815X

[19] Furnell S., Magklaras G., Papadaki M., Dowland P. (2001), ‘A Generic Taxonomy for Intrusion Specification and Response’, Proceedings of Euromedia 2001, Valencia, Spain, pages: 125-131.

[20] Tuglular T. (2000), “A preliminary Structural Approach to Insider Computer Misuse Incidents’, EICAR 2000 Best Paper Proceedings: pages 105-125.

[21] Aslam T., Krsul I., Spafford E.(1996), “Use of a Taxonomy of Security Faults”, Technical Report TR-96-051, COAST Laboratory, Department of Computer Sciences, Purdue University, IN, 1996.

[22] Bach M. (1986), 'The design of the UNIX Operating System', Prentice Hall International Editions, NJ, 1986. Page 10 defines the term “process” in an operating system context.

[23] Richter J. (1997), 'Advanced Windows', Microsoft Press, Redmond, Washington.

[24] Papadaki M., Magklaras G., Furnel S., Alayed A. (2001), 'Security vulnerabilities and System Intrusions – The need for Automatic Response Frameworks', Proceedings of the 2001 Small Systems Security International Conference, Las Vegas, USA, September 2001.
[25] Moore, D., Voelker, G. and Savage S. (2001), "Inferring Internet Denial of Service Activity", Proceedings of the 10th USENIX Security Symposium, Washington D.C, August 2001, pp. 9-22.
[26] Frykholm, N. (2000), Countermeasures against Buffer Overflow Attacks, White Paper, RSA Laboratories.
[27] Postel, J. and Reynolds, J. (1983), TELNET Protocol Specification, Request For Comments (RFC) 854, IETF Network Working Group, May 1983.
[28] Ylonen, T. (1995), The SSH (Secure Shell) Remote Login Protocol, Internet Draft, IETF Network Working Group, 15 November 1995, Available http://www.free.lp.se/fish/rfc.txt
[29] Ziegler, R. (2002), LINUX Firewalls, Second Edition, New Riders Publishing, Chapter 2, pp. 48-50.
[30] Sharda, N. (1999), Multimedia Information Networking, Prentice Hall Inc., Chapter 12.
[31] Wood B. (2000). "An insider threat Model for Adversary Simulation", SRI International, Research on Mitigating the Insider Threat to Information Systems - #2: Proceedings of a Workshop Held by RAND, August 2000.
[32] Schultz, E.E. (2002). "A framework for understanding and predicting insider attacks", Computers & Security, vol. 21, no. 6, pp. 526-531.
[33] Shaw E.D., Ruby K.G., Post J.M. (1998), "The Insider Threat to Information Systems", Security Awareness Bulletin, No. 2/98, Political Psychology Associates Ltd.
[34] Brancik K.C.(2008),"Insider Computer Fraud An in-depth Framework for Detecting and Defending Against Insider IT Attacks", Auerbach Publications, Taylor & Francis Group, ISBN 1-4200-4659-4.
[35] Magklaras G., Furnell S. (2005), "A Preliminary Model of End User Sophistication for Insider Threat Prediction in IT Systems", Computers & Security, Volume 24, Issue 5, August 2005, Pages 371-380.
[36] Magklaras, G. (2005), An Architecture for Insider Misuse Threat Prediction in IT Systems, MPhil Thesis, School of Computing, Communications and Electronics, University of Plymouth, UK.
[37] NSTISSAM. (1999), The Insider Threat To US Government Information Systems, NSTISSAM INFOSEC /1-99, U.S. National Security Telecommunications And Information Systems Security Committee, Available http://www.cnss.gov/Assets/pdf/nstissam_infosec_1-99.pdf.
[38] Cappelli D., Moore A., Shimeall T.J., Trzeciak R. (2006). "Common Sense Guide to Prevention and Detection of Insider Threats", 2nd Edition Version 2.1, Carnegie Mellon University Cylab, URL:http://www.cert.org/archive/pdf/CommonSenseInsiderThreatsV2.1-1-070118.pdf
[39] AccessData Inc. web portal, URL: http://www.accessdata.com/
[40]Guidance Software Inc. web portal, URL:http://www.guidancesoftware.com/computer-forensics-ediscovery-software-digital-evidence.htm
[41] Hay B., Nance K., Bishop M. (2009), "Live Analysis Progress and Challenges", IEEE Security & Privacy, Volume 7, Number 2, pages 30-37.
[42] Adelstein F. (2006), "Live Forensics: Diagnosing Your System without Killing it First", Comm. ACM, vol.49, no.2, 2006, pages 63-66.
[43] The Computer Online Forensic Evidence Extractor (COFEE), Microsoft Corp.
URL: http://www.microsoft.com/industry/government/solutions/cofee/default.aspx

[44] Petroni N., Walters A., Fraser T., and Arbaugh W.(2006), "FATKit: A Framework for the Extraction and Analysis of Digital Forensic Data from Volatile System Memory" ,Digital Investigation Journal 3(4):197-210, December 2006.

[45] Carrier B.D.(2006), “Risks of Live Digital Forensic Analysis”, Comm. ACM, vol. 49, no. 2, 2006, pages 56-61.

[46] Spinellis D. (2001), “Notable design patterns for domain-specific languages”, The Journal of Systems and Software Volume 56, Issue 1 (2001), pages 91-99.

[47] Consel, C. (2004), “From A Program Family To A Domain-Specific Language”, in Lengauer, C.; Batory, D.; Consel, C.; Odersky, M. (Eds.), Domain-Specific Program Generation, Lecture Notes in Computer Science 3016, Springer-Verlag, pp. 19-29.

[48] Raymond E.S. (2003) “The Art of UNIX Programming”, 1st Edition, Addison-Wesley Professional.

[49] Fowler M. (2005). “Language Workbenches: The Killer-App for Domain Specific Languages?”, URL: http://martinfowler.com/articles/languageWorkbench.html#ProsAndConsOfLanguageOrientedProgramming

[50] The association of LISP users portal, URL: http://www.lisp.org/alu/home

[51] The Smalltalk Programming Language Portal, URL: http://www.smalltalk.org/main/

[52] The Ruby Programming Language portal, URL: http://www.ruby-lang.org/en/

[53] Ousterhout J. (1998). “Scripting: Higher Level Programming for the 21st Century”, IEEE Computer magazine, March 1998 issue, pages pp. 23-30.

[54] The W3C Extensible Markup Language web page: URL:http://www.w3.org/XML/

Towards an Insider Threat Prediction Specification Language

G.B.Magklaras[1], S.M.Furnell[1] and P.J.Brooke[2]

[1] Network Research Group, School of Computing, Communications & Electronics, University of Plymouth, Plymouth, United Kingdom
[2] School of Computing, University of Teesside, Middlesbrough, United Kingdom

nrg@plymouth.ac.uk

Design/Methodology/Approach

Various information security surveys indicate that misuse by legitimate (insider) users has serious implications for the health of IT environments. A brief discussion of survey data and insider threat concepts is followed by an overview of existing research efforts to mitigate this particular problem. None of the existing insider threat mitigation frameworks provide facilities for systematically describing the elements of misuse incidents, and thus all threat mitigation frameworks could benefit from the existence of a domain specific language for describing legitimate user actions. The paper presents a language development methodology which centres upon ways to abstract the insider threat domain and approaches to encode the abstracted information into language semantics.

Research limitations/implications

Due to lack of suitable insider case repositories, and the fact that most insider misuse frameworks have not been extensively implemented in practice, the aforementioned language construction methodology is based upon observed information security survey trends and the study of existing insider threat and intrusion specification frameworks. The development of a domain specific language goes through various stages of refinement that might eventually contradict these preliminary findings.

Practical implications

This paper summarizes the picture of the insider threat in IT infrastructures and provides a useful reference for insider threat modeling researchers by indicating ways to abstract insider threats. The problems of constructing insider threat signatures and utilizing them in insider threat models are also discussed.

Keywords: Intrusion Detection, Insider Threat, Insider Misuse, Domain Specific Language, Intrusion Specification

Introduction

The Information Security world often focuses on analyzing and counteracting threats of external origin. However, the problem of insider IT misuse is also an existing headache for the health of IT infrastructures. Surveys published by the British Department of Trade and Industry (DTI) and PriceWaterhouseCoopers (PWC, 2004) as well as the San Francisco-based Computer Security Institute (Richardson, 2003) are good sources for getting a qualitative and quantitative feeling of computer security incidents. Relevant information derived from these surveys is presented in the following section.

Amongst the various research and development issues related to the process of mitigating the problem of internal threats lies the ability to describe the actions that constitute the elements of the threat in a consistent manner. This goal can be achieved by constructing a suitable Insider Threat Prediction Specification Language (ITPSL), in order to facilitate ways of standardizing the description of Insider IT Misuse incidents and thus aid tools designed for detecting and preventing them.

After introducing the problem by quoting incident statistics, basic terminology and discussing some of the Insider IT misuse mitigation frameworks, this paper focuses on the methodology required to construct the language itself, by examining ways to classify insider incidents, as well as the suitability of pre-existing intrusion specification schemes for insider threat specification.

The Insider IT Misuse Threat

An ‘insider’ is a person that has been legitimately given the capability of accessing one or many components of the IT infrastructure, by interacting with one or more authentication mechanisms (plain text password, PKI, biometric or smart card token). The word ‘legitimately’ is a key term, as it emphasizes the main difference between an insider and an external cracker. An insider should always be able to have at least a point of entry in one or more computer systems. The implications of having such a point of entry is that an insider does not usually need to consume as much time and effort to obtain additional privileges as an external cracker does, in order to exploit IT infrastructure vulnerabilities and mount an attack. It also means that an insider is less likely to get caught by implemented security measures because of the level of trust that she enjoys.

The other side of the insider IT misuse problem relates to what can be considered as misuse activity. Although the great majority of the people are familiar with the generic meaning of the word 'misuse', when we try to map it to an insider IT context, there is a need to clarify certain issues.

Insider IT misuse can be a very subjective term. In fact, one of the most challenging tasks is to draw a clear line that separates an IT misuser from a person that uses the available resources in an acceptable way and for an approved purpose. The words 'acceptable' and 'approved' imply the presence of rules that qualify (or quantify) conditions of allowable usage for the resources concerned. These rules are often embodied within an IT usage policy. Part of this organisation-wide policy is the information security policy, defined as the '*set of laws, rules, practices, norms and fashions that regulate how an organisation manages, protects, and distributes the sensitive information and that regulates how an organisation protects system services*' (Caelli et al. 1991).

After defining the terms ‘insider’ and ‘misuse’, we also need to discuss the context of the term ‘threat’. Pfleeger et al. (2003) define the term threat in an IT infrastructure context as “a set of circumstances that has the potential to cause loss or harm”. As a result, in legitimate user context, these circumstances might involve intentional IT misuse activities such as targeted information theft, introducing or accessing inappropriate material, and accidental misuse (e.g. unintentional information leak). In addition, there is also potential for flaws in the design and implementation of the computer system, which could render it susceptible to insider misuse. Such flaws may include improper filesystem permissions or relaxed information security policies, and in conventional information security parlance these are termed vulnerabilities.

The widespread manifestation of insider IT misuse incidents is an existing problem for the health of IT infrastructures. The DTI/PWC 2004 survey (PWC, 2004) mentions that that Insider Misuse has doubled since the year 2002, mainly driven by the increased adoption of World Wide Web and Internet related technologies. The same survey shows that the gap between insider and outsider incidents is smaller for respondents of medium and large scale organizations (Figure 1). The

CSI/FBI survey (Richardson, 2003) also indicates closer margins between the occurrence of internal and external incidents (Figure 2).

Take in Figure 1

Take in Figure 2

Magklaras and Furnell (2004) discuss in greater detail the manifestation of the Insider IT Misuse based on the results of a small scale survey.. The survey queried 50 IT professionals of various specialties, including respondents in system administrator, IT security and management roles. Despite the small number of respondents, their majority had a technical background on various aspects of the insider IT misuse, providing an insight on notable trends of the problem and thus establishing a profile of a misuser.

In particular, some important highlights of Magklaras and Furnell (2004) include the fact that the three most frequent types of IT misuse for the respondents of the survey were the downloading of pornographic material, the abuse of email resources, and the theft or malicious alteration of data. In direct comparison, the DTI/PWC 2004 survey (PWC, 2004) highlights the incidents of web browsing misuse, misuse of email, and unauthorized access to systems or data as the major system misuse categories (Figure 3).

In addition, all of the professionals surveyed by Magklaras and Furnell (2004) indicated some preference towards the existence of certain pre-employment security checks for prospective employees. The DTI/PWC survey (PWC, 2004) indicated that the majority (66%) of the respondents usually perform some sort of general employee background check during the recruiting stage, checking. This survey also comments that the absence of these security checks from company procedures is clearly a serious omission.

Finally, the profile of an insider threat was also refined by indicating that sophisticated users are more likely to misuse an IT infrastructure than less IT-literate users (Magklaras and Furnell, 2004).

The frequency of occurrence is not the only indicator of the impact of insider incidents. There are also substantial financial costs attributed to legitimate user actions. However, due to a combination of factors (such as the smaller percentage of respondents willing to state financial losses in information security surveys and the way some of the surveys associate stated losses to incident types) the process of safely estimating true insider costs is rendered infeasible.

Take in Figure 3

The various survey results combine to suggest that internal incidents are here to stay and their mitigation should be a priority issue for IT professionals.

Insider IT misuse mitigation frameworks

There are numerous research and development efforts that attempt to address the problem of legitimate user misuse. All of them focus on predicting or sensing insider threats. The process of predicting a particular set of events in order to prevent their occurrence and provide a better understanding of their underlying mechanisms does not represent a new methodology in the field of science. The utilisation of game theory in financial forecasting (Gibbons, 1992), in order to predict the value of shares in the stock exchange market and the processing of seismic data for oil discovery purposes (Helbig, 1993) are notable examples of methodologies that already serve our world and used on a daily basis by analysts, as value-added tools that help their activities.

The insider IT misuse mitigation framework suggested by Wood (2000) is one of the earliest examples of qualifying a set of metrics addressing the insider threat. The framework suggests that a malicious insider can be qualified in terms of distinct attributes such as the amount of access she has on some part or all parts of the IT infrastructure (physical and logical access in terms of privileges), her level of familiarity with the internal workings of the target systems, her motives as well as the skills, tactics and processes she uses to mount an attack.

A more recent research effort by Schultz (2002) presents a preliminary framework for understanding and predicting insider attacks by providing a combination of behavioural and system usage related metrics. The paper mentions the detection of system usage patterns that may act as “signatures” of a legitimate user or certain indicators of an attack preparation (“deliberate markers” and “preparatory behaviour”). Sequences of actions that might not be detected in individual systems, but which could certainly indicate misuse when considered against multiple systems are discussed. There is also a discussion of aspects of a legitimate user’s personality that could serve as threat indicators. In particular, on-line (e-mail, IRC or other forms of computerised human-to-human communication) verbal behaviour with signs of aggression, dominance towards particular people might serve as a good prognosis factor of certain attacks (“verbal behaviour”).

Magklaras and Furnell (2002) discuss an alternative framework for insider IT misuse mitigation, focusing on insider threat metrics that could be collected at system level. The Insider Threat Prediction Model (ITPM) is a three-level hierarchy of mathematical functions evaluated in a bottom-up approach. Each of the threat component functions models particular aspects of insider attributes and behavior. The end result is an integer, the Evaluated Potential Threat (EPT), which classifies the level of potential threat a particular user represents for the IT infrastructure and could be used as an indicator of whether the user poses a threat or not.

While each of the aforementioned frameworks has is its own theoretical advantages and disadvantages, they also have something important in common. Whether one places emphasis on verbal behaviour, gauging of user knowledge, or the observation of user action sequences in order to sense or predict insider threat, all of these metrics could benefit from a standardized way of describing them efficiently, in order to make insider IT misuse threat signatures. This is the subject of the following sections.

Insider Threat Prediction Specification Language: The need and its construction methodology

Information security surveys and mass media might report accurately the outline of an insider misuse case. They do not provide a complete picture about the exact conditions under which the incident occurs, nor they always reveal in detail the sequence of user actions. As an example, one should consider the high-profile case of Robert Hanssen (CNN, 2002). A 56-year old trusted FBI veteran, Hanssen abused his trusted access to the Automated Case Support System that contained classified information about ongoing investigations and handed critical information to Russian agencies. In return, he was receiving large sums of money, inflicting a great deal of damage upon the prestigious image of the Federal Bureau of Investigation and the national security of his country.

The motives and the outline of Hanssen's methodology were covered by the mass media. Some of the details related to his data hiding techniques were also mentioned in computing sites (Slashdot, 2001). However, even the later details are not enough for someone to re-construct the case in a laboratory for the purposes of experimenting and developing insider threat prediction techniques. If one is not a member of the forensic specialist team that handles an insider misuse case, he will be able to only speculate about the actions and the attack path a malicious insider had followed. This creates a lack of suitable case repositories noted by (NSTISSAM, 1999), and is one of the limiting factors in the field of insider IT misuse mitigation research.

The establishment of a world-wide insider case repository would be of great aid to researchers. However, apart from the coordination, the building of such a repository would require a way to unambiguously describe the insider misuse actions in a standard manner. This paves the way for the shaping of a Domain Specific Language (DSL), a semantic mechanism tailored specifically for describing the details of a particular task. The main goal is the usage of appropriate semantics to reduce the effort required to reference and manipulate elements of that particular domain.

DSL schemata have been employed successfully in a number of different areas. Consel (2004) discusses the range of applications that have employed a DSL which includes device driver construction, active networking and operating system process scheduling. This list is by no means exhaustive and it really concerns all domains that consist of software entities that have enough common elements to be considered as a whole. A DSL is really a framework that offers the ability of building specific and concise notations to express a problem domain, as well as safe (as predictable) code due to semantic restrictions. Both of these properties are very desirable in the process of developing insider threat specifications.

Thus, a methodology for deriving a Domain Specific Language includes three important steps:

- the abstraction of the domain, which involves the removal of all the unnecessary details of the environment;
- the systematic categorisation of the necessary (abstracted) details into language semantics;
- the process of engineering the developed semantics into software.

The derivation of the necessary abstractions is achieved partly by the establishment of the aforementioned insider threat mitigation frameworks. These frameworks rely on two important elements that achieve the abstraction. The first is a careful classification of the Insider IT threat elements. The classification schemes (or taxonomies) enhance the ability to examine the problem in a more systematic way, could certainly form the core of a specification language and are a common occurrence in the Information Security literature. The second element is a model that combines all the threat elements and provides an estimate for the magnitude of the threat.

Whilst the models are discussed in detail by various researchers (Wood, 2000; Schultz, 2002; and Magklaras and Furnell, 2002), the section that follows will focus on the taxonomy issue. The formation of a structured way to identify insider IT misuse threat elements forms a key component of a threat specification language. The language semantics and the process of engineering them into software will also be discussed in later sections.

A taxonomy for Insider Misuse Threat Prediction

An overview of intrusion specification taxonomies is provided by Furnell et al. (2001). Amongst these taxonomies, one that specifically addresses insider IT misuse incidents is given by Tuglular (2000). This taxonomy integrates an established security policy to the process of classifying computer misuse incidents in three dimensions: incident, response and consequences. These dimensions can be divided into additional sub-dimensions that further classify a particular misfeasor. Tuglular's paper is one of the first to suggest a 'target-type of threat' association as a way to prevent insider misuse. The target is an 'asset' and the rule is called a 'strategy' in the terminology he proposes. The suggestion is mentioned in a single sentence and forms the basis for a methodology to predict insider misuse threats. If one can associate successfully certain actions to threats then it establishes the first step towards systematizing insider IT threat prediction.

Most research efforts in the field of intrusion taxonomy classification are still at an early stage. The Tuglular taxonomy, and others mentioned in (Furnell et al. 2001), are useful for the systematic study of intrusions, but they offer little help to a process designed to automatically detect intrusive activities. This is because the classification criteria employed by these taxonomies cannot be qualified or quantified very easily by an Intrusion Detection System with the level of information they exhibit. Moreover, none of these taxonomies is tailored for the process of estimating the likelihood of Insider Threat.

The best way of enhancing the expressiveness of an intrusion taxonomy scheme for insider misuse activities is to focus on the human actions and how their consequences impact the elements of the IT infrastructure that are being targeted. The idea is that it is easier to detect which particular element is affected by a potentially intrusive action, rather than focusing on the task of sensing the motives for initializing an attack.

Another important property of a suitable Insider IT misuse prediction taxonomy is the freedom of the security architect to choose what can be considered as an Insider IT misuse threat indicator. Most taxonomies enforce a rigid framework for classifying phenomena with clear borders of distinction that offer little space for subjective or varying interpretation of facts. This schema does not fit the case of Insider IT misuse prediction. The previous section of this paper argued that there are different views for what is considered as legitimate user misuse amongst the various organizations. Consequently, there are also different views for what is perceived as an insider threat prediction indicator and a taxonomy tailored for the needs of a threat prediction process should be flexible enough to accommodate this fact.

Take in Figure 4

As a result, one can construct a suitable threat prediction taxonomy based around consequences detected at system level. Figure 4 above displays the top level of the taxonomy structure indicating the three primary, non-mutually exclusive levels that address these consequences.

The Operating System (O/S) based consequences are branched down to two sublevels of file-system and memory manipulation, illustrated by Figures 5 and 6 respectively. A justification for this is that a large number of security faults (Aslam et al. 1996) involve filesystem and memory management issues, and indeed the core modules of UNIX (Bach, 1986) and Windows-based (Richter, 1997) operating systems provide (amongst others) specific support for the related functions. Hence, it is safe to assume that these two kernel functional attributes can be used as a strong criterion for further classifying legitimate user activities.

Take in Figure 5

At File/Directory level, a misuser may attempt to read or alter (write/create) certain files. These files might contain sensitive or unauthorised information (information theft or fraudulent modification of vital information). A knowledgeable insider might also attempt to read or modify file information that is not directly related to its content. Bach (Bach, 1986) and Richter (Richter, 1997) emphasize that most Operating Systems allow a file to contain additional information such as access/creation/modification times as well as information that relates the file to its owner and permits access to it under certain conditions. Although the mechanisms that implement these file attributes are different amongst Operating Systems, they are collectively known as file metadata and they are vital mechanisms to secure the privacy, availability and integrity of the file contents. Consequently, they are good candidates for exploitation by a legitimate user who is about to perform a deliberate or accidental misuse act.

The points mentioned in the previous paragraph are also valid for ‘filesystem’ related data. Every Operating System organizes its files and directories by means of a specific set of rules that define how a file (contents and metadata) are about to be stored on the physical medium. The Operating System sub-modules that handle these issues are known as **filesystems**. Attempts to read or alter the physical medium’s Master Boot Record (MBR), intentional or accidental modification of partition table data are some of the most notable auditable actions that could point to legitimate user misuse acts. Robert Hanssen’s case is a classic reminder of this kind of activity. His specially modified 40-track floppy disk was created by a set of filesystem modification actions, in order to create a hidden area to store the sensitive information (Slashdot, 2001).

In addition to filesystem content and metadata modification, a survey of insider misuse conducted by the authors (Magklaras and Furnell, 2004) showed that excessive disk space consumption is perceived as a problem for many of the respondents. Under certain conditions that depend on the configuration of the IT infrastructure, a legitimate user might produce a deliberate or accidental Denial of Service attack (DoS).

Take in Figure 6

In addition to filesystem content and metadata modification, a survey of insider misuse conducted by the authors (Magklaras and Furnell, 2004) showed that excessive disk space consumption is perceived as a problem for many of the respondents. Under certain conditions that depend on the configuration of the IT infrastructure, a legitimate user might produce a deliberate or accidental Denial of Service attack (DoS).

Memory inspection is the best way to see if a legitimate user attempts to run or even install a suspicious program. Indeed, it is one of the core techniques used in the detection of overtly malicious code, such as viruses and Trojan horse programs. However, software threats do not end here, and a problem originating from end-user actions was highlighted by the authors’ aforementioned survey. The majority of the respondents (76%) claimed that an attempt to install one or more unauthorized applications is also classified as a misuse act for their organizations. Hence, this could be used as a strong criterion for the purposes of sensing insider threats in an IT environment. The execution or installation of these programs could be intercepted by either recognizing a program's footprint in memory or by intercepting a well-known series of system calls produced by various suspicious programs. For example, the fact that a non-advanced user is trying to compile an advanced vulnerability scanning tool is an event that should be noticed, and serves as a good indicator of potential misuse activities that are about to follow.

In addition, attempts to consume large memory portions of an operational system that are related to a legitimate user account can serve as good indicators of (intentional or accidental) insider misuse at Operating System level. One might argue that the 'irregular memory usage' sub-categories should really belong under the 'Program execution' hierarchy of events. However, it is possible that someone will produce a quick and easy Denial of Service attack on a running system by forcing the host to commit large portions of system memory to a process, as demonstrated in various case studies described in (Moore et al. 2001). Moreover, a large category of security faults can be achieved by means of accessing normally restricted memory areas, creating what is commonly known as a "buffer overflow" attack (Frykholm, 2000). As a result of these issues, it was felt that a separate sub-category hierarchy should exist to describe these events.

The filesystem and memory manipulation consequences conclude the O/S consequence category of the proposed taxonomy. The next category, "network consequences", represents another distinct set of factors that could be taken into consideration in order to classify insider misuse threat indicators.

Figure 7 illustrates the network-related consequences of acts that could be used as legitimate user threat indicators.

The authors' insider misuse survey indicated that 26% of the surveyed IT professionals consider the content of web pages that a legitimate user visits to be an important Threat Indication factor. Hence, it is reasonable to assume that URLs that contain a 'promising' link to sexually explicit content or to illegal software downloads should be noted as distinct ways of indicating potential to misuse the system (suspicious URLs).

Take in Figure 7

Network packets that are associated with certain legitimate users and indicate the usage of a variety of network protocols and applications that might introduce certain vulnerabilities are also distinct ways of accidental or intentional IT misuse. For example, it could be said that a user that employs the TELNET (Postel and Reynolds, 1983) protocol to login to a multi-user system is more likely to have her account compromised than a user who logins via the Secure Shell (SSH) application (Ylonen, 1995) due to the fact that the earlier application transmits the user password in clear-text form across the network, whereas the latter one encrypts it.

Someone might also like to differentiate between TCP and UDP based applications/protocols. From a potential threat point of view, UDP services are less secure than TCP based ones. For example, Ziegler (2002) discusses in detail how UDP's lack of flow control and state mechanisms can create various data security problems. Consequently, the distinction between the usage of UDP and TCP services can serve as a potential insider misuse threat indicator, on the basis that UDP services are more likely to be accidentally (or intentionally) abused by a legitimate user.

The participants in the authors' survey indicated that resource over-utilization is an existing issue in IT infrastructures. Although the 'Filesystem Manipulation' subcategory of the taxonomy indicates ways with which disk storage capacity can be misused, the results of over-utilisation can also affect network capacity. For instance, a legitimate user could start downloading massive quantities of data, exceeding the network bandwidth cost budget of a business (Downloading over X Mbytes of data in a period Y). The X and Y number limits can be selected by the network administrator according to the company budget requirements.

In addition, a legitimate user might also cause network congestion by exceeding the data network's 'burst' or throughput capacity or exhausting the number of available network endpoints, as described by Sharda (1999). Bandwidth hungry applications, such as video streaming players, and multiple data transfers can cause congestion that can severely impact the performance of a data network or affect the Quality of Service (QoS) of certain applications that require sustained data network throughput.

Finally, incoming or outgoing SMTP headers or attachments might indicate activity related to e-mail misuse that can certainly be traced in network or host level. Outgoing e-mails that contain a set of particular files as attachments (e.g. password database files, other sensitive material) and have unusual destination addresses (e.g. unknown Hotmail accounts, a large number of recipients) should serve not necessarily as intrusion indicators but as insider threat estimators. The plethora of malicious code efforts and phishing techniques may have an external origin, but the threat is realized by the actions of unsuspecting legitimate users. In addition, proprietary information theft could also be realized by means of emailing sensitive material to non-authorized external entities.

The last system consequences category ("hardware") plays an important role in preventing a number of computer system threats. Insiders can often access the physical hardware of the machine very easily. Thus, removal or addition of hardware components, as well as modifications of their default configuration are some of the events that may act as important indicators of insider misuse prediction in a computer system.

From a taxonomy to encoding and language semantics

After identifying and characterizing the insider IT misuse threat factors, the next issue concerns the development of the encoding schemes and semantics of the desired language. Earlier sections made reference to the concept of Domain Specific Languages (Consel, 2004) and the first steps for devising a suitable threat specification language have been made. The Common Intrusion Specification Language (CISL) (Feiertag et al. 1999) consists of a semantic framework to unambiguously describe intrusive activities together with proposed data structures that store the event information and can form standardized messages exchanged by various Intrusion Detection System (IDS) components.

The CISL framework could be re-used for producing a suitable Insider Threat Prediction Specification Language. However, the framework would require substantial re-engineering to achieve this goal. The existing CISL framework and the latest related research are summarized in the paragraphs that follow. The discussion then proceeds to present the CISL major flaws from an insider threat specification perspective, and suggests an approach to overcome these problems.

In CISL, the semantic representation of intrusive activities is achieved by the formation of an S-Expression. This is a recursive grouping of tags and data, delimited by parentheses. The tags provide semantic clues to the interpretation of the S-Expression and the data might represent system entities or attributes. For this reason, the tags are also called Semantic Identifiers (SIDs).

The best of way of illustrating how CISL works is by considering an example. The statement (Hostname ‘frigg.uio.no’) is a simple S-Expression. It groups two terms, without semantically binding them. One can guess that it refers to a computer system with the FQDN name ‘frigg.uio.no’, but the true meaning of the statement is still vague. In fact, the full semantic meaning of S-Expressions becomes apparent when one forms more complex S-Expressions, by means of combining several SIDs into a sentence.

Figure 8 illustrates a suitably crafted CISL intrusion specification which could be translated in the following plain English translation:

“*On the 24th of February 2005, three actions took place in sequence in the host ‘frigg.uio.no’. First, someone logged into the account named 'tom' (real name ‘Tom Attacker’) from a host with FQDN 'outside.firewall.com'. Then, about a half-minute later, this same person deleted the file*

'/etc/passwd' of the host. Finally, about four-and-a-half minutes later, a user attempted but failed to log in to the account 'ksimpson' at 'frigg.uio.no'. The attempted login was initiated by a user at 'hostb.uib.no'."

The particular CISL sentence describes a malicious attack that erases an important system file of a UNIX system and consists of three multi-SID S-Expressions. In general, a sentence can be formed by one or more S-Expressions nested at different levels. However, there are strict rules that allow the nesting of S-Expressions. The rules are defined by the nature of the SIDs, as there are several different types of them.

Take in Figure 8

Verb SID's are joined together in a sentence by conjunction SIDs. In the previous example of Figure 8, 'And' is the conjunction SID that holds together the three SIDs that form the sentence. In addition, a CISL sentence might employ role, adverb, attribute, referent and atom SID types. Role SIDs indicate what part an entity plays in a sentence (such as 'Initiator'). Adverb SIDs provide the space and time context of a verb SID. Attribute SIDs indicate special properties or relations amongst the sentence entities, whereas atom SIDs specialise in defining values that are bound to certain event instances (for instance 'Username'). Lastly, referent SIDs allow the linking of two or more parts of a sentence ('Refer to' and 'Refer as'). There are additional SID types, but the aforementioned ones are the most commonly employed.

One can clearly observe a structural hierarchy for forming complex sentences that also contributes to the semantic meaning. This semantic structure is inspired by the syntax of natural languages. A verb is always at the heart of every sentence and is followed by a sequence of one or more qualifiers that describe the various entities that play parts in the sentence, or qualify the verb itself. In addition, a similar hierarchy is also reflected in the formation of the previously described insider misuse taxonomy.

CISL (Feiertag et al. 1999) is not only about semantic rules. Its authors were concerned with the encapsulation of the structured semantic information into the 'Generalised Intrusion Detection Object' (GIDO), data structures that hold the encoded event information. The purpose of encoding the information in a standard way is to make the process of exchanging the information amongst various IDS components easy.

The Common Intrusion Detection Framework (CIDF) that embodies CISL (Feiertag et al. 1999) considers an IDS as a group of discrete functional components that exchange messages.. Some of the components intercept an intrusion event (E-boxes) or organise them into searchable collections (D-boxes), whereas others analyze it (A-boxes) to determine whether the event is worth looking and event take some sort of action (R-boxes). One of the major objectives behind this conceptual IDS view was to enable seamless integration that accommodates for inevitable differences in IDS implementations. This is another important issue that concerns the formation of an ITPSL.

Unfortunately, despite the well-conceived interoperability target, the CISL GIDO encoding process introduced many problems. Doyle (1999) has criticized many of the aspects of the CISL GIDO structure. Although the purpose of the document was to evaluate the fitness of CISL for use in the DARPA Cyber Command and Control (CC2) initiative, the paper identifies serious inadequacies that concern the CISL time resolution data representation facilities, as well as data throughput limitations caused by the fixed size of the GIDO data structure. Finally, Doyle comments on the lack of support for the next generation Internet Protocol (Version 6). Whilst these points are fair, they could easily be corrected by making the necessary changes to the relevant data types and overcome the perceived obstacles. In fact, section 7 of the CISL standard (Feiertag et al. 1999) contains specific guidelines that explain how to add information to a GIDO, to clarify or correct its contents. This suggests that the encoding principles are certainly extensible.

A more serious aspect of Doyle's critique in (Doyle, 1999) refers to the semantic structure of the CISL language. In particular, his criticism that CISL has "no facilities for representing trends or other complex behavioral patterns; ill-specified, inexpressive, and essentially meaningless facilities for representing decision-theoretic information about probabilities and utilities" indicates that the language would be a bad choice for describing information about a threat prediction model. The basic reasoning behind this critique is that CISL is too report-orientated and threat mitigation requires a different level of information, not just mere report structures of what is happening on one or more systems. These indeed represent more serious limitations that would require a more radical re-design of the CISL.

In response to the CISL encoding limitations, the IETF Intrusion Detection Exchange Format working group (see www.ietf.org/idwg) took over the scope of the CIDF work. It addressed most of the GIDO encoding issues by introducing a new Object Oriented format for encoding and transmitting Intrusion Detection related information. The Intrusion Detection Message Exchange Format (IDMEF) (Curry et al. 2004) enriched the type of standardized information that IDS sensors may represent, as well as the process of standardizing the exchange of messages using protocols such as IDXP (Feinstein et al. 2004) and data exchange languages such as XML (W3C, 2006). For example, the IDMEF ‘Confidence’ and ‘Impact’ classes can now be used to represent decision theoretic information (Curry et al. 2004). The earlier can assign a confidence and thus a probability to an observed event, whereas the latter relates privilege escalation consequences to three broad severity levels. This functionality can serve as the basis for encoding probabilistic information, in order to use it in an ITPSL concept.

These standardization features were lacking from the previous CIDF platform and they constitute a very important step towards establishing better interoperability amongst different IDS products. However, at the time of writing, the working group has not managed to expand on the semantic scope the CISL language and address the various expressiveness issues that Doyle mentioned. The IDMEF draft standard (Curry et al. 2004) proposes more extensive encoding and data structures, but it does not suggest semantic guidelines like the ones proposed by the CIDF framework. For IDMEF, the term ‘language’ refers to the data types and encoding principles for IDS data and not to the syntactical guidelines of an Intrusion Specification Language.

Hence, the establishment of an Intrusion Specification Language tailored to Insider Threat Prediction could be achieved by adopting the basic syntactic guidelines of the CISL framework and address the syntactic inadequacies indicated by Doyle (1999). After the semantic refinement step, an effort to match the suggested event expression statements to the IDMEF data structures should take place. This will ensure that the ITPSL scheme would be fully compliant with the relevant IETF standards of the research field.

Figure 9 below illustrates the process of turning an ITSPL-based text description into a multi-level threat signature. A high-level text description of the threat is parsed by a suitably crafted compiler and turned into network, file and memory-level (multi-level) statements that detect the different threat components at system level. The produced signature could then populate a database of signatures, such as the one Magklaras (Magklaras, 2005) proposes for the Insider Threat Prediction Model (ITPM). This process could also facilitate the building of a world-wide case repository of insider threat cases, such as the one mentioned in (NSTISSAM, 1999). This would benefit computer security analysts and forensic specialists as well as IDS vendors.

Take in Figure 9

The process of refining the original CISL semantic schema would enrich the original language by adding new atom and adverb SID types that represent decision-theoretic and probabilistic information. For example, user privileges related to authorized or not network, file and memory level operations can be represented by the IDMEF ‘Impact’ class. In addition, there are plenty of IDMEF data structures that can represent information related to the file, network and command execution ITPSL expression components. The ‘FileList’ and ‘FileAccess’ classes contain adequate attributes to represent the file attributes. The ‘Address’ class can represent network related data, and lastly, the ‘Process’ class could accommodate most of the requirements of the command execution data of the ITPSL schema.

Such a language would help one to establish more easily insider threat signatures that could be used in various IDS implementations and computing architectures. Figure 10 illustrates how the language interacts with the ITPM model (Magklaras and Furnell, 2002). The ITPSL encoded threat signature is fed into a module that translates its contents to Operating System specific Application Programming Interface (API) directives. Each OS/Computing platform implements different mechanisms to facilitate the monitoring of filesystem, network and memory related events. The translation of the ITPSL encoded statements to platform specific instructions achieves the desired platform independence feature. The monitoring modules feed the ITPM model with the necessary data, in order to establish whether a user constitutes a threat with respect to the signature contents. Whilst the ITPM is shown here to interact with ITPSL, the scheme could also prove useful to other threat modeling efforts. The produced positive or negative result could then be used by an IDS or IPS system, in order to further increase (or reduce) the intensity of monitoring various operational aspects of a system or react to prevent/block intrusive activity respectively.

Take in Figure 10

For instance, let us consider the hypothetical case of a malicious insider stealing proprietary information and forwarding it to a rival company. Assuming that the misuser gets caught, a security specialist normally gathers forensic evidence from the computing infrastructure. He might look at the media used to transfer information, the information access patterns, the contents of emails and personal storage media. He could then establish the ITPSL text level description of the incident on a repository database. A researcher or IDS vendor product engineer could then acquire the posted signature, recreate the misuse threat and see how he could improve the detection of the threat at different stages and computer architecture levels (network, file and memory level). He could also refine the signature, in order to include undiscovered variations of the incident, as the language framework should provide a good way to structure insider threat information. The produced signatures could then be re-used in future systems to model and warn about eminent threats of similar nature.

ITPSL in comparison to currently available security tools

Earlier sections of the paper discussed the lack of facilities for systematically describing the elements of misuse incidents in current threat mitigation frameworks. Nevertheless, there is currently a variety of tools that help IT practitioners monitor and respond to insider activities. The variety of commercial and open source solutions is too large to include an exhaustive discussion of all available tools here. However, a comparison of the proposed language features and the currently employed IT security tools will indicate where this research effort fits in the overall field of practice.

Internet firewalls (Zwicky et al. 2000) are commonly employed tools looking closely at data passing through today's networked IT infrastructures. There are many types of firewalling mechanisms, ranging from stateless and stateful packet filtering to more sophisticated application-aware network filters. Irrespective of the mechanism employed, the basic idea is that network traffic is inspected at protocol and possible payload level in search of patterns or trends that indicate malicious traffic. Although firewalls were traditionally employed to protect an IT infrastructure from attacks of external origin, they are currently utilized to block traffic from the inside to the external world and in that respect they can act as mechanisms to mitigate insider threats. In fact, most of the networking consequences of the proposed taxonomy (Figure 7) could be mapped to firewall toolkit rules.

Intrusion Detection and Prevention Systems (IDS/IPS) are some of the latest tools that provision more refined mechanisms to detect and prevent an information security breach (Endorf et al. 2003). IPS devices exercise access control mechanisms to protect computer systems from malicious acts.

They were originally developed in an attempt to increase the accuracy of passive network monitoring techniques and provided large improvements over the aforementioned firewall mechanisms. IPS devices could be viewed as extensions of IDS mechanisms. IDS are devices designed not only to prevent and (where possible) respond to a plethora of computer security incidents, but also to integrate the operation of other security components (anti-virus, firewall and cryptographic applications) into one overall system. The implications of this integrated approach are that IDS/IPS approaches examine both host and network based data to mitigate threats.

The File-system manipulation O/S consequences (Figure 5) as well as the Memory Manipulation O/S consequences (Figure 6) of the proposed taxonomy are typical examples of concepts that are today directly applicable to IDS/IPS solutions targeting insider attack vectors. Thus, one might wonder about the necessity of the proposed language. If most of the proposed detection criteria of the taxonomy that abstracts the problem are already employed in available solutions today, what is the need for yet another language? The answer to this question lies in how the signatures are encoded and how easy it is for a security administrator to encode a scenario using current security tools targeting insider incidents. Firewalls, IDS/IPS, antivirus and anti-spyware solutions have rule writing conventions that could to some extent be viewed as mini DSL constructs. Examples of these rule writing conventions are the IPTABLES firewalling rules (Ziegler, 2002) and Sourcefire’s SNORT rule parser engines (Beale et al. 2003), which are widely employed to encode intrusion signatures for their IPS/IDS product series. Similar examples can be found on other firewall and IPS/IDS product offerings, as well as antivirus solutions. In fact, anti-virus vendors construct the signatures and offer them as part of their product, with their customer not engaging at all in any stage of the virus signature construction.

The common traits of today's available solutions indicate proprietary coding schemes or schemes that require a substantial amount of system-specific level of knowledge, in order to encode a threat signature, with evident cross-vendor boundaries. A threat signature from product vendor A will generally not be usable with the product of vendor B, and when it is, the effort and the compatibility nightmares will always make the task of porting signatures an undesirable overhead. This is exactly where ITPSL fits into the picture. It can provide not only the means for constructing a structured repository of insider misuse cases but also act as a complement of IDS/IPS and other frameworks or tools (Figure 10), providing a high level 'glue' to describe insider threat components. This component could be used by commercial vendors not only as an information repository but also as a tool that eases the porting of signatures and scenarios to their product platform.

Conclusions

Insider Threat is a problem that affects the well being of IT infrastructures. Various frameworks for mitigating insider misuse exist following different philosophies of approaching the problem. However, all frameworks lack a way of describing precisely acts of legitimate user misuse, an important ability for every researcher in the field. A domain specific language tailored around insider misuse incidents can facilitate this need and enhance the capabilities of these frameworks.

Although the paper has presented the concept of the language, the development of the proposed approach is currently a work in progress. As such, it would be premature to attempt to convey more specific details at this stage. Indeed, constant refinement of the semantics and language interface mechanisms is expected, especially during the early stages of its development. An important first step in the process of constructing such a language is the abstraction of the problem domain by means of classifying insider misuse incidents. Insider taxonomies are frequently encountered in the research literature. However, the building of an insider misuse language requires a threat taxonomy based on consequences detected at system level. This design approach would allow the language to fit easily around events that can be captured in an automated fashion and not on parameters that need to be deduced such as motive, for example.

The proposed taxonomy could then pave the way for encoding threat signatures. The CISL (Feiertag et al. 1999) and the IDMEF (Curry et al. 2004) frameworks are examples of previous research attempts to provide standardized semantics for specifying intrusions, as well as ways to encode intrusion specific information. By adapting their semantics and data structures to the field of insider misuse, one could produce a mechanism to encode insider threat specific information and make use of it in insider threat modeling frameworks.

References

Aslam, T., Krsul, I. and Spafford E. (1996), Use of a Taxonomy of Security Faults, Technical Report TR-96-051, COAST Laboratory, Department of Computer Sciences, Purdue University, IN.

Beale, J., Foster, J.C. and Posluns, J. (2003), Snort 2.0 Intrusion Detection, Syngress Publishing, ISBN: 1931836744

Bach, M. (1986), The design of the UNIX Operating System, Prentice Hall International Editions, NJ.

Caelli, W., Longley, D. and Shain, M. (1991), Information Security Handbook, Stockton Press.

CNN.com (2002), "The case against Robert Hanssen", In-Depth Special series. Available http://edition.cnn.com/SPECIALS/2001/hanssen/

Consel, C. (2004), "From A Program Family To A Domain-Specific Language", in Lengauer, C.; Batory, D.; Consel, C.; Odersky, M. (Eds.), Domain-Specific Program Generation, Lecture Notes in Computer Science 3016, Springer-Verlag, pp. 19-29.

Curry, D., Debar, H. and Feinstein, B. (2004), The Intrusion Detection Message Exchange Format, Internet Draft, Intrusion Detection Exchange Format working group, Internet Engineering Task Force, 8 July 2004.

Doyle, J. (1999), Some representational limitations of the Common Intrusion Specification Language, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1999 Revision

Endorf, C., Schultz, E. and Mellander, J. (2003), Intrusion Detection and Prevention: The Authorative Guide to Detecting Malicious Activity, McGraw Hill, ISBN: 0072229543.

Feiertag, R., Kahn, C., Porras, P., Schnackenberg, D., Staniford-Chen, S. and Tung, B. (1999), A Common Intrusion Specification Language (CISL), 11 June 1999, Available http://www.isi.edu/~brian/cidf/drafts/language.txt

Feinstein, B., Matthews, G. and White, J. (2002), The Intrusion Detection Exchange Protocol (IDXP), Internet Draft, Intrusion Detection Exchange Format working group, Internet Engineering Task Force, 22 April 2003, Available http://www.ietf.org/internet-drafts/draft-ietf-idwg-beep-idxp-07.txt.

Frykholm, N. (2000), Countermeasures against Buffer Overflow Attacks, White Paper, RSA Laboratories.

Furnell, S., Magklaras, G., Papadaki, M. and Dowland, P. (2001), “A Generic Taxonomy for Intrusion Specification and Response”, Proceedings of Euromedia 2001, Valencia, Spain, 18-20 April 2001, pp. 125-131.

Gibbons, R. (1992), A Primer in Game Theory, Prentice Hall International.

Helbig K. (1993), Modelling the Earth for Oil Exploration: Final Report of the CEC's Geoscience I Program 1990-1993, Pergamon Publishing.

Magklaras, G. (2005), An Architecture for Insider Misuse Threat Prediction in IT Systems, MPhil Thesis, School of Computing, Communications and Electronics, University of Plymouth, UK.

Magklaras, G. and Furnell, S. (2002), “Insider Threat Prediction Tool: Evaluating the probability of IT misuse”, Computers & Security, Vol 21, No 1, pp. 62-73.

Magklaras, G. and Furnell, S. (2004), "The Insider Misuse Threat Survey: Investigating IT misuse from legitimate users", *Proceedings of the 5th Australian Information Warfare & Security Conference*, Perth Western Australia, 25-26 November 2004, pp. 42-51.

Moore, D., Voelker, G. and Savage S. (2001), "Inferring Internet Denial of Service Activity", Proceedings of the 10th USENIX Security Symposium, Washington D.C, August 2001, pp. 9-22.

NSTISSAM. (1999), The Insider Threat To US Government Information Systems, NSTISSAM INFOSEC /1-99, U.S. National Security Telecommunications And Information Systems Security Committee, Available http://www.cnss.gov/Assets/pdf/nstissam_infosec_1-99.pdf.

Pfleeger, C. and Pfleeger, S. (2003), Security in Computing, Third Edition, Prentice Hall.

Postel, J. and Reynolds, J. (1983), TELNET Protocol Specification, Request For Comments (RFC) 854, IETF Network Working Group, May 1983.

PWC. (2004), Information Security Breaches Survey 2004 – Technical Report, Available http://www.pwc.com/images/gx/eng/about/svcs/grms/2004Technical_Report.pdf

Richardson, R. (2003), 2003 CSI/FBI Computer Crime and Security Survey, Computer Security Institute. Spring 2003.

Richter, J. (1997), Advanced Windows, Microsoft Press, Redmond, Washington.

Schultz, E.E. (2002), “A framework for understanding and predicting insider attacks”, Computers & Security, Vol 21, No 6, pp. 526-531.

Sharda, N. (1999), Multimedia Information Networking, Prentice Hall Inc., Chapter 12.

Slashdot (2001), “Spying and Technology: Robert Philip Hanssen”, posting on Slashdot.org, 22 February 2001, Available http://slashdot.org/articles/01/02/22/0622249.shtml

Tuglular. T. (2000), “A preliminary Structural Approach to Insider Computer Misuse Incidents”, EICAR 2000 Best Paper Proceedings: pp. 105-125.

W3C. (2006), Extensible Markup Language, Architecture Domain, World Wide Web Consortium (W3C), Available http://www.w3.org/XML/

Wood, B. (2000), An insider threat Model for Adversary Simulation, SRI International, Research on Mitigating the Insider Threat to Information Systems - #2: Proceedings of a Workshop Held by RAND, August 2000.

Ylonen, T. (1995), The SSH (Secure Shell) Remote Login Protocol, Internet Draft, IETF Network Working Group, 15 November 1995, Available http://www.free.lp.se/fish/rfc.txt

Ziegler, R. (2002), LINUX Firewalls, Second Edition, New Riders Publishing, Chapter 2, pp. 48-50.

Zwicky, E.D., Cooper, S. and Chapman, D.B. (2000), Building Internet Firewalls, Second Edition, O'Reilly & Associates, ISBN: 1565928717

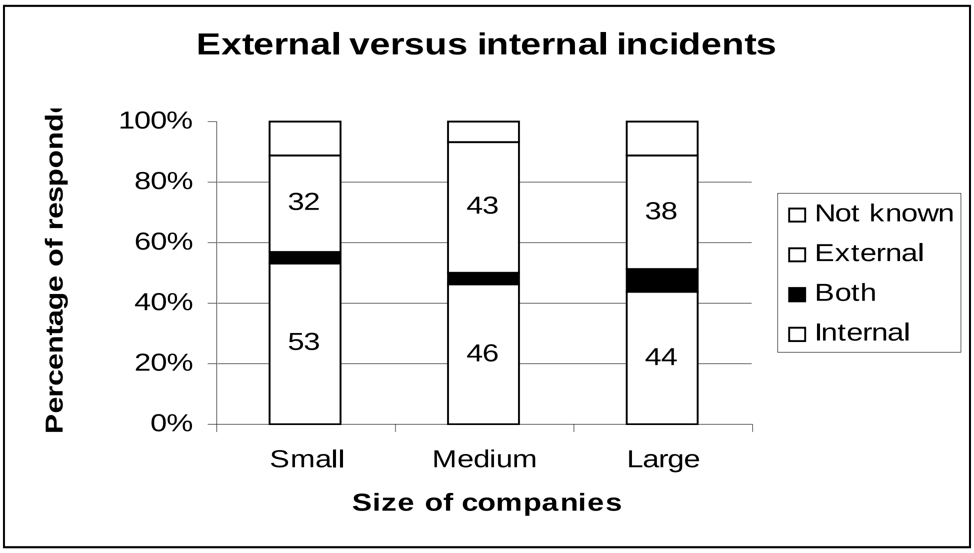


Figure 1: External versus internal incidents in terms of report frequency (PWC, 2004)

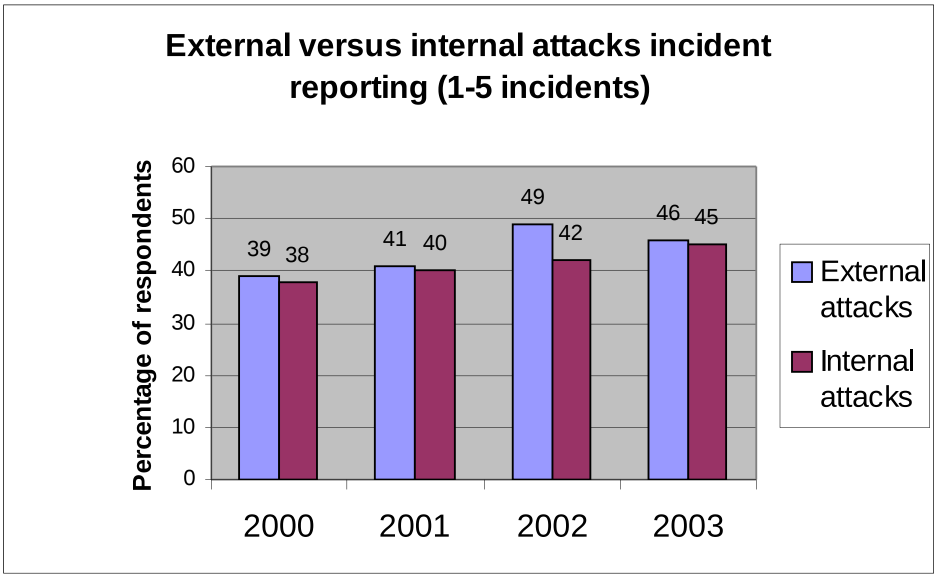


Figure 2: External versus internal attack incident frequency (Richardson, 2003)

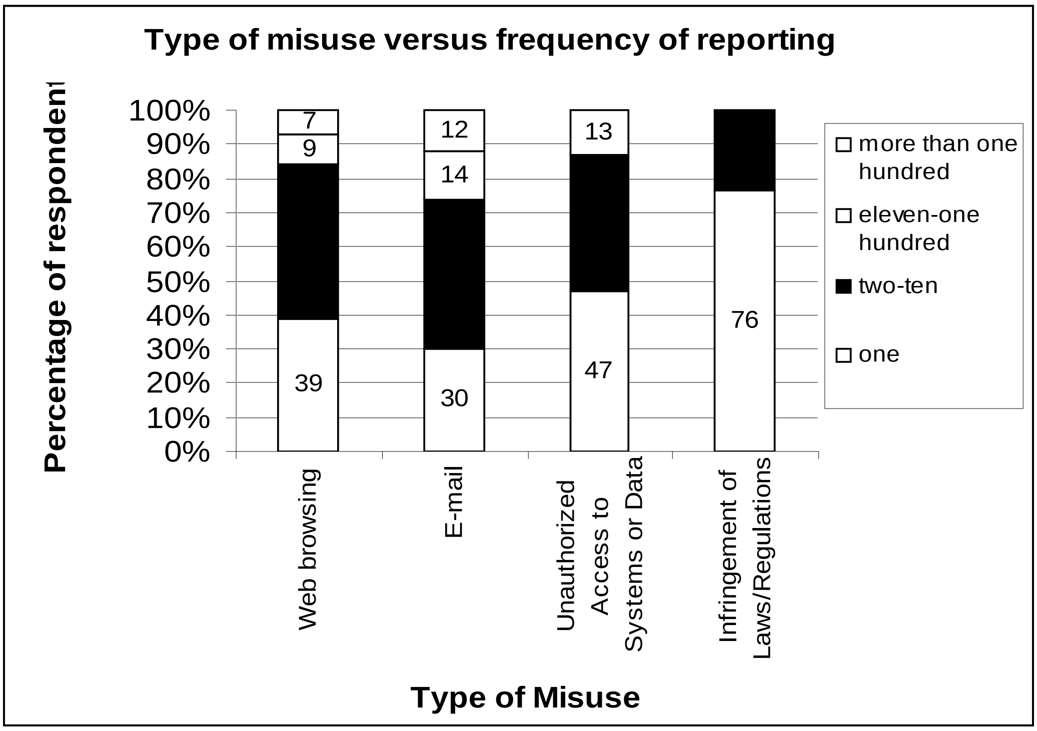


Figure 3: Types of misuse reported by UK businesses (PWC, 2004)

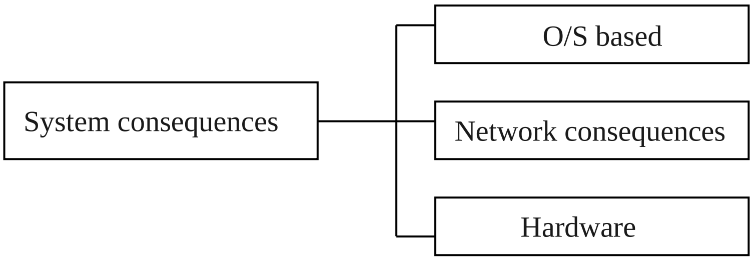


Figure 4: Top level of an insider threat prediction taxonomy

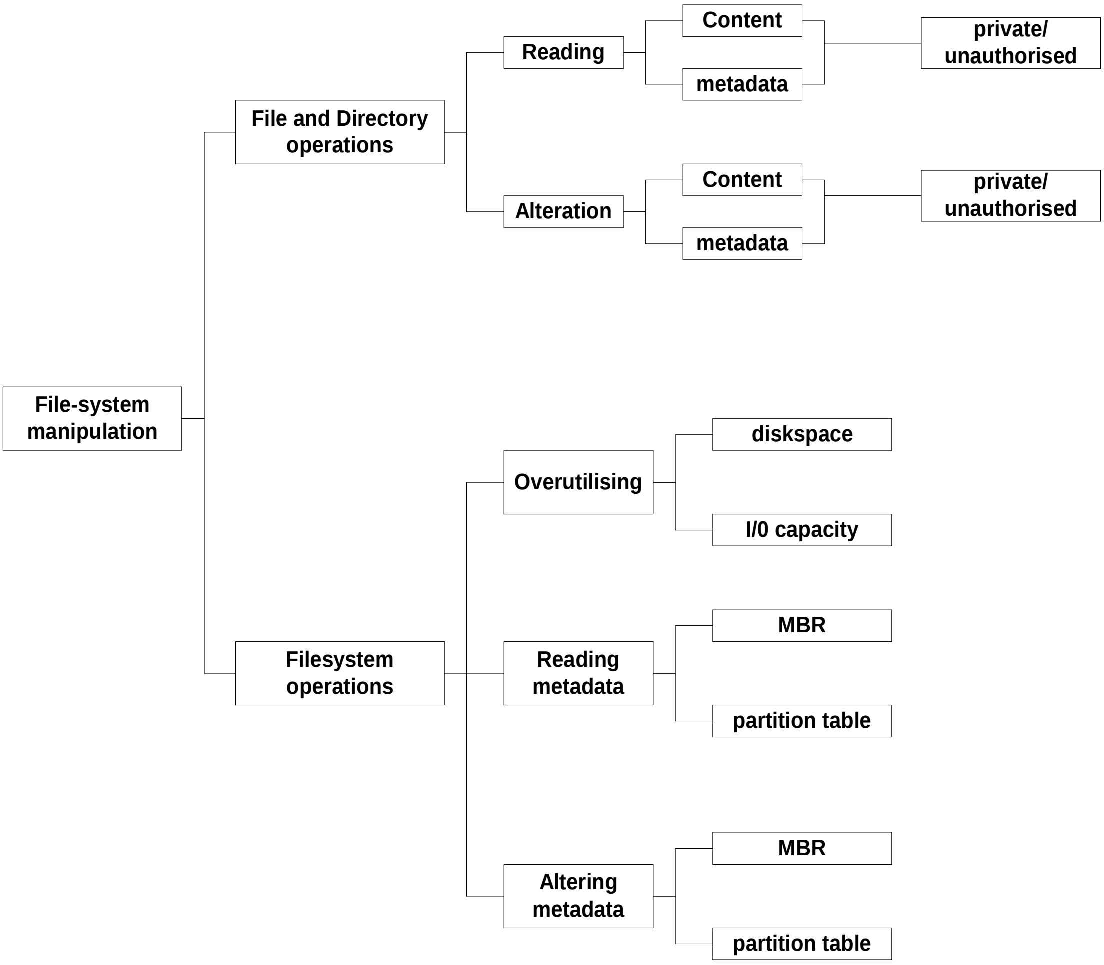


Figure 5: File-system manipulation O/S consequences

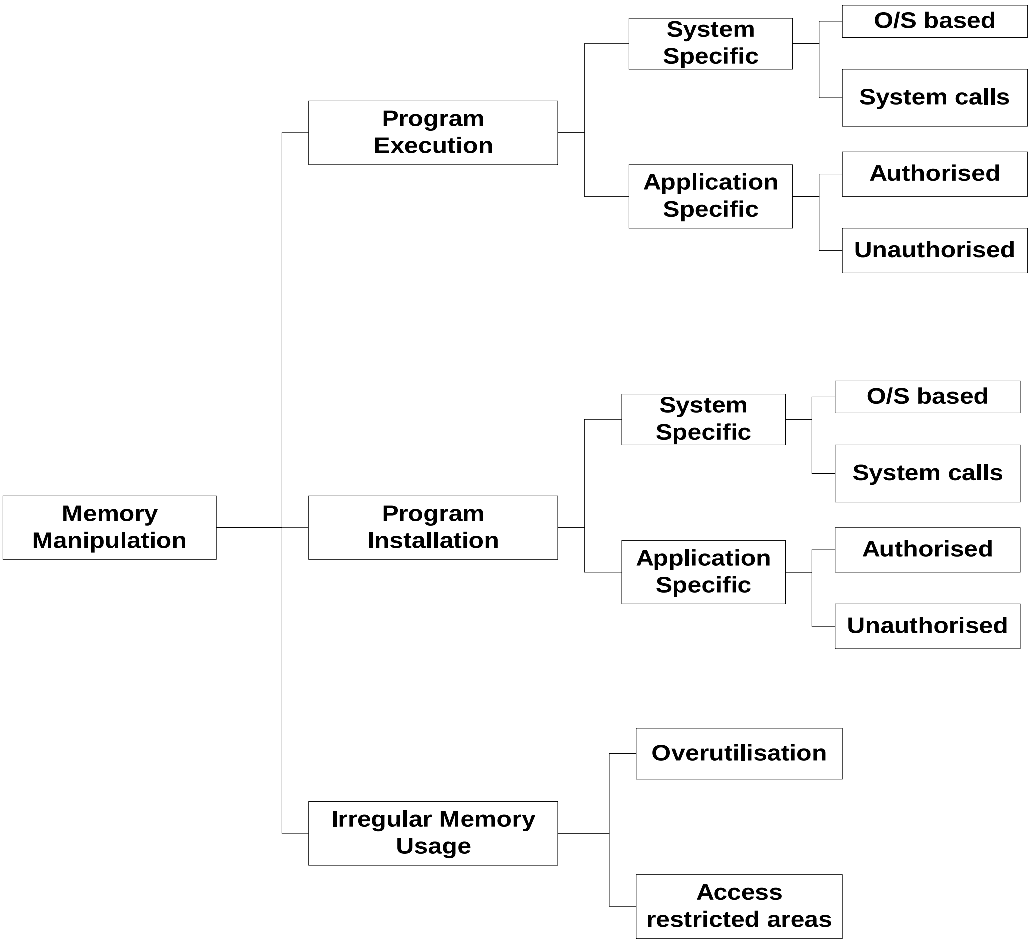


Figure 6: Memory Manipulation O/S Consequences

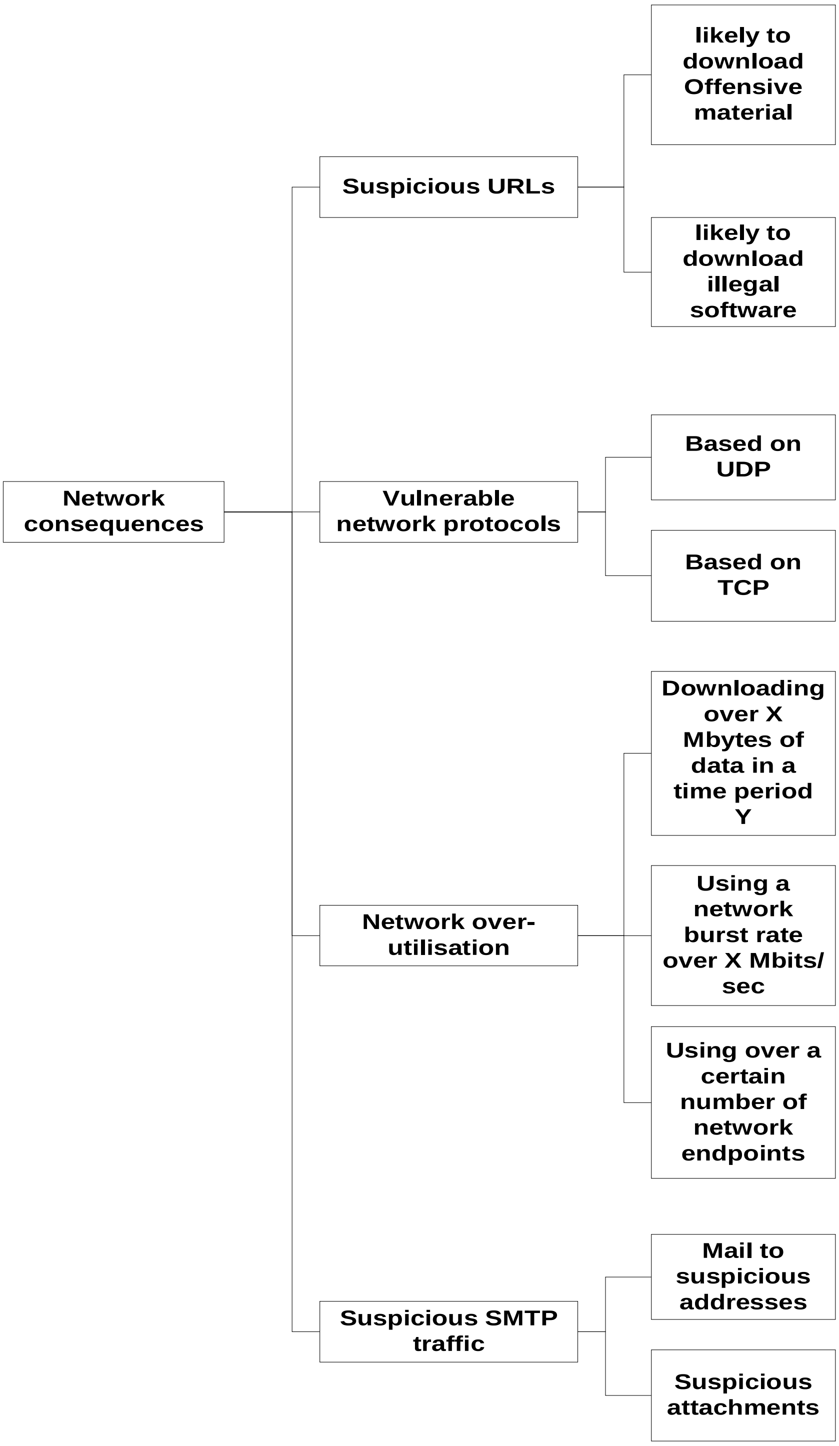

Network consequences
Suspicious URLs
likely to download Offensive material
likely to download illegal software
Vulnerable network protocols
Based on UDP
Based on TCP
Network over-utilisation
Downloading over X Mbytes of data in a time period Y
Using a network burst rate over X Mbits/ sec
Using over a certain number of network endpoints
Suspicious SMTP traffic
Mail to suspicious addresses
Suspicious attachments

Figure 7: Network consequences of the insider IT misuse prediction taxonomy

```
(And

        (OpenApplicationSession
            (When
                (Time 14:57:36 24 Feb 2005)
            )
            (Initiator
                (HostName 'outside.firewall.com')
            )
            (Account
                (UserName 'tom')
                (RealName 'Tom Attacker')
                (HostName 'frigg.uio.no')
                (ReferAs 0x12345678)
            )
            (Receiver
                (StandardTCPPort 22)
            )
        )
        (Delete
            (World Unix)
            (When
                (Time 14:58:12 24 Feb 2005)
            )
            (Initiator
                (ReferTo 0x12345678)
            )
            (FileSource
                (HostName 'frigg.uio.no')
                (FullFileName '/etc/passwd')
            )
        )
        (OpenApplicationSession
            (World Unix)
            (Outcome
                (CIDFReturnCode failed)
                (Comment '/etc/passwd missing')
            )
            (When
                (Time 15:02:48 24 Feb 2005)
            )
            (Initiator
                (HostName 'hostb.uib.no')
            )
            (Account
                (UserName 'ksimpson')
                (RealName 'Karen Simpson')
                (HostName 'frigg.uio.no')
            )
            (Receiver
                (StandardTCPPort 22)
            )
        )
    )
```

Figure 8: CISL sentence syntax example

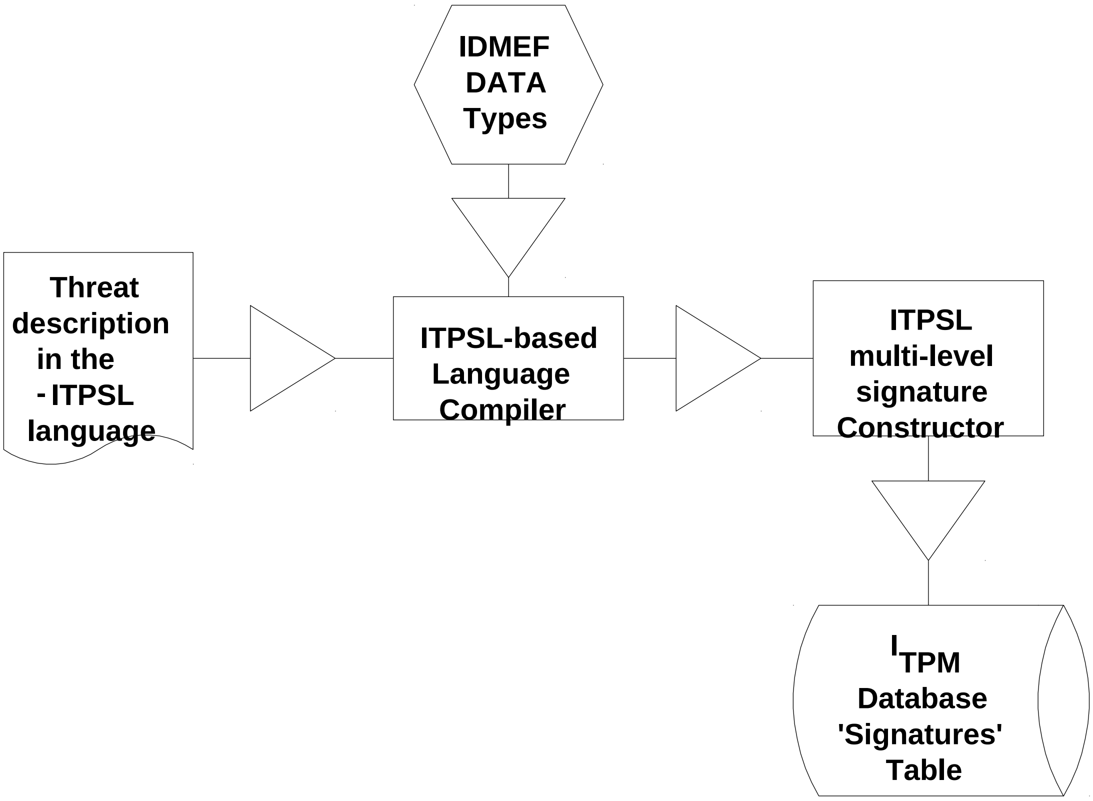


Figure 9: From ITPSL text description to a threat signature

Figure 10: ITPSL/ITPM relationship