A FRAMEWORK FOR CORRELATION AND AGGREGATION OF SECURITY ALERTS IN COMMUNICATION NETWORKS

F. ALSERHANI

PhD

UNIVERSITY OF BRADFORD

2011

A FRAMEWORK FOR CORRELATION AND AGGREGATION OF SECURITY ALERTS IN COMMUNICATION NETWORKS

A reasoning correlation and aggregation approach to detect multi-stage attack scenarios using elementary alerts generated by Network Intrusion Detection Systems (NIDS) for a global security perspective

FAEIZ ALSERHANI

Submitted for the Degree of

Doctor of Philosophy

School of Computing, Informatics and Media

University of Bradford

2011

Acknowledgements

It takes a long time to complete a PhD thesis, though not as long as it takes to lay some rail track, surprisingly. I would here like to express my thanks to the people who have been very helpful to me during the time it took me to write this thesis. First and foremost, I would like to thank my supervisors, Dr. Andrea Cullen and Prof. Irfan Awan, for the time and effort they put into helping me along the way. Their insight was extremely valuable and I have learned much from their feedback. The good advice, support and friendship of Prof. Irfan has been invaluable on both an academic and a personal level, for which I am extremely grateful. I also would like to show my gratitude to the people of Syphan technologies, Mr. Pravin Merchandi, Dr. Jules Dissos and Mr. Shakeel Ali , for their comments and support during the initial phase of the research.

The years spent in Bradford would not have been as wonderful without my colleague Mr. Monis Akhlaq. I am sincerely and heartily grateful to Monis for his personal and scientific support which has made a great impact on completion of this work. We have spent unforgettable times in our lab discussing issues and solving problems related to our research. I also would like to thank my colleague Dr. Rasha Osman for her assistance and interesting feedback during the research years.

Finally, I am forever indebted to my parents and my family for their understanding, endless patience and encouragement when it was most required. They have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

Abstract

The tremendous increase in usage and complexity of modern communication and network systems connected to the Internet, places demands upon security management to protect organisations' sensitive data and resources from malicious intrusion. Malicious attacks by intruders and hackers exploit flaws and weakness points in deployed systems through several sophisticated techniques that cannot be prevented by traditional measures, such as user authentication, access controls and firewalls. Consequently, automated detection and timely response systems are urgently needed to detect abnormal activities by monitoring network traffic and system events. Network Intrusion Detection Systems (NIDS) and Network Intrusion Prevention Systems (NIPS) are technologies that inspect traffic and diagnose system behaviour to provide improved attack protection.

The current implementation of intrusion detection systems (commercial and open-source) lacks the scalability to support the massive increase in network speed, the emergence of new protocols and services. Multi-giga networks have become a standard installation posing the NIDS to be susceptible to resource exhaustion attacks. The research focuses on two distinct problems for the NIDS: missing alerts due to packet loss as a result of NIDS performance limitations; and the huge volumes of generated alerts by the NIDS overwhelming the security analyst which makes event observation tedious.

A methodology for analysing alerts using a proposed framework for alert correlation has been presented to provide the security operator with a global view of the security perspective. Missed alerts are recovered implicitly using a contextual technique to detect multi-stage attack scenarios. This is based on the assumption that the most serious intrusions consist of relevant steps that temporally ordered. The pre- and post-condition approach is used to identify the logical relations among low level alerts. The alerts are aggregated, verified using vulnerability modelling, and correlated to construct multi-stage attacks. A number of algorithms have been proposed in this research to support the functionality of our framework including: alert correlation, alert aggregation and graph reduction. These algorithms have been implemented in a tool called Multi-stage Attack Recognition System (MARS) consisting of a collection of integrated components. The system has been evaluated using a series of experiments and using different data sets i.e. publicly available datasets and data sets collected using real-life experiments. The results show that our approach can effectively detect multi-stage attacks. The false positive rates are reduced due to implementation of the vulnerability and target host information.

Figures

Tables

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Introduction

This recent era has witnessed a massive growth in the use of computer network applications. More hosts are connected to the Internet to speed up business processes and to provide more accessibility. This has increased reliance on e-business paradigms providing dynamic and complex environments with interconnections of critical infrastructure elements. The inherently invisible nature of Internet usage, in most cases due to political reasons and the absence of legislation, has made these systems targets for hackers and intruders [1]. Traditionally, firewalls have been used as perimeter guards for organizational networks to filter incoming and outgoing traffic [2]. However, the number of sophisticated attack methods is growing, such as multi-vector, multi-stage and insider attacks, in addition to data leakage threats as more sensitive data is stored in open-mode networks. Hence, an extra layer of defence is needed for deep packet inspection and context-aware detection.

1.2 Security status

In spite of the existence of security mechanisms, incidents of attacks are still occurring because attackers make use of flaws in implemented applications and services [3]. There are plenty of methods for bypassing traditional security systems, such as buffer overflow, application layer attacks to trick users, and insider threats. Most of these behaviours are considered legitimate because they do not violate the applied security policies, though they are in fact malicious. In addition, from a business point of view, a trade-off has to be made between strict security policies and productivity [4].

To provide protection mechanisms against the new trends in intrusion techniques, advanced and intelligent intrusion detection and protection systems are required [5].

Firewalls and software patches can no longer be regarded as reliable means of providing a defence against well-defined and novel attacks. Network traffic data has to be inspected and analysed in depth in order to detect malicious behaviour. Stateless analysis relying on packet-level observation does not improve the efficiency of the protection systems. Furthermore, different data sources and incorporated detection techniques have to be used in order to achieve higher level protective systems.

In this respect, Network Intrusion Detection Systems (NIDSs) have been proposed as complementary security tools providing sensors to observe network traffic for any malicious activities. Several approaches with different capabilities have been developed to achieve this functionality, such as signature-based and anomaly-based mechanisms. Pre-defined attack patterns are supplied to signature-based approaches to detect any matching between these patterns and the received traffic data. In anomaly-based mechanisms, generated *normal profiles* are compared with the incoming activities to judge abnormalities. However, the common purpose of NIDSs is to detect potential intrusions in network traffic, generating security alarms. NIDSs can perform in detective mode or proactive mode, but immediate response may affect the usability of the protected systems, particularly if alarms are false.

Recent advances in CPU processing power, memory and network speed have "stressed" the performance of NIDSs [6]. The difference between advances in networking speed and processing speed has created what has been called a *performance gap*, because communication speed has developed far in advance of processing speed [7]. This has imposed challenges for NIDSs, as they have to process multi-Giga traffic inline. Several methods have been developed for load balancing, distributed sensors [8] and parallel processing, but another challenge has emerged in the coordination of these sub-system units.

1.3 The limitations of NIDSs

NIDSs can be considered a second line of defence in the protection of production networks, and they can cooperate with firewalls and antivirus systems to achieve maximal protection coverage. Both research communities and commercial vendors have been working for several years to improve the functionality of NIDSs. However, these systems still suffer from limitations that can generally be summarized as follow:

1) High volume of generated low-level alerts [9], which makes it impractical for human analysts to pursue such an amount of information. Even worse, the quality of the observed data varies between certain intrusions and activities with a low degree of confidence. Typically, NIDSs produce alerts which are mapped to atomic detected events, but are not capable of determining to which incidents the detected alerts belong. The administrator has to analyse the data manually or use simple analysis tools based on statistical methods. Moreover, hostile actions are assumed to be infrequent compared to legitimate activities, so the analysis of a large amount of data in order to observe rare information is a cumbersome.

2) A high rate of false positives is a major limitation of NIDSs and one that makes their effectiveness questionable. [10] states that more than 99% of the alerts generated by NIDSs are false positives. False positives are produced because the NIDS believes, based on its detection mechanism, that the detected activity is malicious. This weakness is mainly due to the fact that the system is unable to precisely determine the ultimate goal of the intrusion. Essentially, there is only a slight distinction between legitimate and malicious behaviour, as even malicious behaviour makes use of the facilities offered by the target system. For instance, in signature-based methods, there must be a balance between the level of specificity and the generality of a signature. A very

specific definition keeps the rate of false positives to a minimum, whereas general signatures broaden the detection space but increase the false positive rate.

3) A high rate of false negatives is also another critical issue, where the NIDS is not able to detect malicious behaviour. That is due to the unavailability of pattern descriptions in signature-based methods or the fact that the behaviour is similar to a normal one in anomaly-based methods. However, skilful attackers use known attacks but combine them with available evasion techniques [11, 12] to deceive the NIDS and to pass undetected. Moreover, 0-day attacks are not identified, as they are unknown and their definitions are unavailable to the NIDS.

4) The difficulty in handling a huge amount of traffic packets, diverse network protocols and sophisticated Web services. Deep inspection and comprehensive analysis have to be done for higher-degree detection and protection. And that requires massive processing capabilities and intelligent algorithms. The typical deployment of a NIDS is at the network edge, where the aggregation of organizational traffic passes. In inline mode, this has made the achievement of acceptable connectivity without any latency a challenge. Many approaches have been developed to cope with these problems, such as traffic splitting [13, 14] across a number of sensors to balance the load. Other techniques involve shifting from software-based to hardware-based solutions [15-17].

5) The sophistication and complexity of modern attacks exploiting new emerging services, such as Web application technologies [18]. Simple attacks to violate security policies are no longer used, particularly after years of security patches to protect core systems. Current trends in intrusion techniques are to employ hidden attacks that are difficult to be recognised by traditional security means. Multi steps of normal-type activities incorporate an attack and after breaking into the system, the intruder remains

silent for the longest possible time. Identifying this type of behaviour is not a straightforward matter without intensive observation and behavioural analysis. Most implementations of NIDSs, both commercial [19] and open source [20, 21], rely on stateless signature-based methodologies. Basic statistical approaches are implemented to detect anomalous behaviour using anomaly-based methods. Moreover, NIDSs need to be supplied with enough information from network traffic and from the end systems [12] to obtain the full picture of the protected systems. Such cooperation between security systems rarely exists and is still in a developmental phase. Efficient correlation techniques have to be implemented in order to differentiate between benign and malicious behaviour.

6) Scalability to support the points mentioned above, as networks nowadays are changing in respect to bandwidth and diversity of services available. The implementation of NIDSs may be sufficient for a certain time, but they need adaptive mechanisms in order to react to different situations.

7) Testing NIDSs to evaluate their operations is cumbersome [22]. There are no efficient approved methodologies to evaluate such systems due to the complexity of NIDS and the operational environments in which they are deployed. Ad hoc approaches have been developed and will be discussed in detail in Chapter 3.

8) Statefulness analysis in order to build an accurate behaviour profile remains a stressing demand for NIDSs. For instance, Snort [23, 24] performs analyses on a connection basis only, so the need for higher levels of context analysis is crucial. This is based on the assumption that each occurring event may be connected with other events, and the correlation is useful in understanding the target of the event in question. Several NIDS claim they perform stateful analyses, but the concept of this type of analysis is

sometimes unclear. Stateful analysis does not only consist of performing TCP reassembly or IP de-fragmentation, but also the analysis of the semantic of multiple activities, including levels of connection, applications and services.

9) Evasion techniques [11, 12, 25, 26] have been used to exploit the implementation ambiguities of protocols and services. Moreover, the gap between application developers and security experts has led to the production of programs with bugs exploited creatively by hackers. Malicious data distributed over fragmented packets to confuse detection systems or session slicing are examples of such evasion methods, or it can also take the form of obfuscation of Web application requests to break into vulnerable applications.

1.4 Alert correlation systems

Principally, Intrusion Detection Systems (IDSs) in general are useful only if their detection results are reviewed and analysed to derive current system security. Some difficulties affecting IDS operations have been stated, and to alleviate some of these limitations alert management systems have been proposed. Alert correlation systems as complementary tools deployed in a typical scenario separately from IDS, as the latter are performance sensitive [27]. The objective of these approaches is to receive alert streams from the IDS, create logical relationships between alerts, link each alert to its related contextual information, and provide a high-level view of the system's security situation. In prime, the receiving audit data is obtained from various IDS so it is used in alert correlation process. However, alert correlation can be also applied on individual IDSs to detect coordinated attacks and to reduce alarm volumes. It is worth mentioning that alert correlation is not an isolated process, and that several components are involved in achieving correlation, aggregation, alert reduction and alert verification.

It has been identified in the cyber security field that well-planned attacks consist of a number of stages conducted in a temporal order. True alerts belonging to intrusions generated by the IDS are not isolated; they also reflect the sequential pattern of the attacker. However, IDSs consider these alerts as individual events and report this to the administrator with a huge amount of alerts, most of them false positives or ones not critical to the protected system. A high-level view of these incidents can assist in recognizing the attacker's plan and taking rapid action to protect the network. Moreover, IDSs, due to their limitations, cannot detect all variations of unseen attacks. However, alert correlation systems can predict the upcoming attack based on the pervious behaviours of attackers. Also, false alarms can be excluded because they are often of isolated and non-critical events.

As a motivating example of a multi-stage attack, the Botnet attack scenario is considered as follows: the attacker performs scanning activities looking for a vulnerable host in a target network in order to install a backdoor. The IDS can detect the scanning behaviour, rating it as a low-risk activity, and also detects the shellcode installation but it is not as a part of the Botnet attack. Then the infected machine sends a connection request to the C&C (command-and-control) server in order to download the configuration file, which is typically encrypted. The IDS in this case can detect the URL of the C&C server as a blacklist. Note that the second phase does not necessarily need to be linked to the first phase, particularly if they occur far away from each other. The second stage can pass undetected using some obfuscation techniques; however, the server response containing some abnormal data in HTML format is detected. After that, maintenance and update activities are performed by downloading some binaries. The infected machine consequently performs a fast scan for other machines and sends a large number of DNS requests. Hence, if these stages are treated individually, they may

be considered isolated activities with low priority. Alert correlation systems process the resulting alerts to discover the connection between them based on causal relationships and to provide a global picture for the administrator.

Alert correlation systems are intended to fill the semantic gap between high-level abstracted events and low-level elementary alerts. The security administrator's requirements include: reduction of data redundancy, intelligent correlation of IDS alerts, recognition of attack scenarios, and a visualised attack scene. To achieve these tasks, different correlation mechanisms are employed, including alert similarities [28-30], attack scenario specifications [31], pre- and post-conditions [32-35], and data-mining techniques [36-38]. These mechanisms vary in their requirements and inner workings, but their common function is to build an abstracted knowledge about different attacks.

Despite several efforts made to achieve the objectives of alert correlation systems, only a limited part of the correlation function has been addressed. Correlation tasks cannot be implemented alone, but require some other cooperative system components, such as aggregation, verification and data reduction. It has been mentioned that the main motivation behind the notion of alert correlation is to identify the connection between alerts. However this task, without removing data redundancy, will make it more complex and the information size will be increased considerably. In addition, the practice of correlation is processing-intensive and the typical deployment is connected to the IDS. It is impractical to rely on a single component for a complex function such as alert correlation; instead, a framework consisting of various components should be used. Each sub-system is responsible for certain tasks and all system parts are integrated in a systematic manner.

1.5 Motivation

With the rapid advances in communication networks and the increase in the number of incidents of detected attacks [39], NIDSs have become a major component of security systems. However, NIDS have two major problems: first, missed attacks due to unknown attack patterns or because packets carrying attack evidence are dropped due to performance limitations. Second, the huge volume of irrelevant alerts overwhelming security analysts makes event observation tedious. This thesis has addressed these two practical problems through two phases:

1) NIDS (software-based) evaluations in high-speed environments to characterise the problem of missed alerts caused by packet loss.

2) Alert correlation systems to mitigate the two previous problems using a contextual recovery technique that provides the security analyst with a global view of the security perspective.

The motivation behind this work inspired from the two phases above can be summarized as:

a) Performance evaluation of NIDSs (software-based) in high-speed networks: The typical deployment of software-based NIDSs is installation on a dedicated server with minimum active services. This setup is quite susceptible to resource-exhaustion attacks, especially in high-speed environments. Sending a large amount of traffic or using computationally expensive techniques like fragmentation can compromise a NIDS or make it start dropping packets. Few efforts have been made to measure the performance of NIDSs, and most of the evaluation methodologies are based on moderate traffic flow [40]. This is because generating traffic in high volumes requires a sophisticated test-bench, which is not always available to most researchers. A test-bench has been built in

our lab using various machines and switches to simulate real-life network traffic. In addition, the evaluation of NIDSs is elusive and there is no typical methodology to test, as few vendors [41] offer it and it is not available to researchers.

b) Alerts missed by NIDSs: As mentioned above, NIDSs may miss some alerts due to unavailable attack descriptions or packet loss in Gig networks. The missing of such alerts is very dangerous, as serious attacks can pass undetected. Several works have been carried out to deal with this issue [27, 42] and to characterise NIDS performance. NIDS vendors recommend the application of conservative engine detection configurations to minimise resource consumption. This can affect the effectiveness of NIDSs as the detection space may be narrowed. Other efforts have been made to distribute traffic making use of balancers [13, 14, 43]; however, these may add extra complexities. The implementation of NIDS on hardware is potentially the optimal solution for this issue [16, 44, 45]. However, hardware is expensive, difficult to configure and tedious to maintain. In addition, the problem of missed alerts caused by a lack of signatures will not be alleviated. For this reason, recovery techniques are needed to reason about missed alerts, whether solely or contextually.

c) Overwhelming administrators with irrelevant alerts: Typically, IDSs continuously generate vast amounts of alerts, and most of them are either false or low-level risk alerts. These data have to be analysed to obtain security status. This flood of information may end up hiding serious activities that could end up being overlooked. Simple analysis tools based on statistics provide certain details but do not reduce the resulting data. Hence, a mechanism needs to be devised to reduce alert flooding without losing critical details focusing on serious and coordinated activities.

d) False positives: It has been identified that approximately 99% of alerts reported by IDS are false positives [10, 46]. This is the result of the reduced quality in the description of current signatures and the imprecise determination of the borderline between legitimate and malicious activities. There are mainly three levels of solution to deal with this issue: 1) at the IDS sensor level, 2) at the protected system level, and 3) at the IDS log level. The first technique is to enhance the IDS detection algorithm to produce a very small number of false alerts. The main focus of these solutions is to build multiple special-purpose IDS [47, 48]. However, this could possibly affect the attack coverage and create compatibility and integration issues. The other two approaches [10, 29, 35, 49-51] are promising in terms of extending the IDS detection domain and focusing on attack-related alerts. Alerts are generated and then post-processed to identify only important information believed to relate to true positives. Vulnerability and protected system information are obtained and supplied to alert correlation systems to identify whether the attack is successful or the alert is a false positive. In addition, the alert correlation system itself performs its functions to discover the relationships between the alerts and aggregate them, ignoring isolated alerts which are most likely false positives.

e) Multi-stage attack recognition: It has been identified in practice [29, 35, 49] that most skilful attacker activities consist of multiple steps (attack scenarios) and occur in a certain time (attack window). An attack is performed using different vectors to gain access to the target system. IDS treat these steps individually, reporting isolated alerts while each step prepares for the next one to complete the intended attack. Identification of such a strategy can lead to the recognition of attack intentions, as well as the prediction of unknown attacks.

f) Slow-and-low attack detection: The new intrusion trend is to be slow [52], while the stages are distributed over a long period of time so as to avoid notice. Another feature is that it is performed with minimum noise, exploiting very small amounts of traffic in order to defeat any anomaly-based technique. Most alert correlation systems, particularly the ones implemented for real time [53], rely on the observation of incoming data during a pre-defined windows size. Memory requirements increase dramatically with the window size and the system becomes a target for state explosion attacks. The only available solution is to remove the detected states from memory in a periodic fashion. This leads to the loss of some of the attack stages if they are temporally diverged. All detected attack phases should be recorded, as the relationship may be discovered after a while.

g) Alert correlation approaches:

- *Algorithms:* The proposed algorithms vary between alert aggregation, data fusion, data reduction and alert correlation. The current trend is to create a cooperative system environment that provides complementary components to achieve practical solutions.

Knowledge base modelling: The core of the correlation systems consists of the supported knowledge bases. Knowledge acquisition methods and the considered features are different, some of them being based on security expert analyses and others relying on pure statistical and machine learning approaches. Knowledge representation plays a major role in the effectiveness of the developed system. The supported data should be formalized in a systematic manner, taking into account specific and general concepts.

Alert verification: One of the main causes of false positives is the knowledge gap between the IDS and the network it protects. The IDS is not capable of identifying the

target system's response after the attack. To bridge this gap, vulnerability, host and network details should be supplied to the correlation system to verify logged alerts. If an alert is assigned low priority, it can be used to extend the attack knowledge without having to consider it a critical element in the attack strategy. Instead of obtaining the target response, which adds more complexity, it is preferable to store an updated knowledge base about the required information.

System implementation to provide a practical ground: The development of required algorithms for alert correlation functions becomes useless if these algorithms are not implemented. The evaluation of the system's effectiveness cannot be carried out without a practical tool. Most proposed approaches have been implemented in an ad hoc manner to show the main functionalities.

Evaluation of alert correlation systems: Generally, the evaluation of IDSs is not an easy task due to the heterogeneous nature of such systems, and alert correlation systems inherit this property. Most evaluation methodologies only focus on a particular part of the system without considering other conditions. Moreover, some researchers validate their work with one or two datasets, some of which do not suit the case. For instance, some datasets consist of attack traffic only [54], which makes the test basic and simple. Others are not originally intended to test alert correlation algorithms. Therefore an intensive evaluation methodology with clear metrics is required, and it needs to be applied to different categories of datasets.

1.6 Contribution

1) Comprehensive performance evaluation of NIDS in a high-speed environment

We have carried out a comprehensive performance evaluation of NIDSs to identify their limitations in high-speed environments. We have designed and implemented a state-of-

the-art, high-speed test lab so as to be able to replicate current and potential threats. This facility has been specifically designed to simulate realistic network traffic conditions comprising different scenarios of background and malicious network traffic. We then evaluated Snort [23], an open-source NIDS, on account of it being a de facto standard. Two broader approaches have been selected to determine the performance of Snort: host-based and virtual-based analyses. This is further supplemented by gauging the performance of the system on different operating system (OS) platforms.

2) A proposed framework for alert correlation

We have proposed a framework for alert correlation consisting of a collection of integrated components to utilize the capabilities of different approaches. This is to formalize a comprehensive solution for correlation, aggregation, data reduction and multi-stage attack recognition. We have presented a Multi-stage Attack Recognition System (MARS) as an alert correlation system to receive alerts from the IDS. The attack scenario is presented as evolving events over time bringing the attack strategy as a graph of connected aggregated phases. The graph explosions in other approaches have been avoided, which typically result in unmanageable attack graphs.

3) Set of proposed algorithms for the framework components:

- *Alert correlation*: We have developed an algorithm for alert correlation functions based on the partial satisfaction of the pre- and post-conditions of each attack. The logical connections are based on hierarchical multilayer specifications of attack capabilities. The correlation is performed for all elementary alerts before aggregation, and then any further correlations can be obtained implicitly for performance purposes.

- *Alert aggregation*: To complement the alert correlation algorithm, an aggregation algorithm has been developed to eliminate data redundancy. The aggregation mechanism assigns a master alert for each group of similar alerts. Thus the main

objective of this algorithm is to minimise the number of nodes in the resulting attack graph. A pre-defined time threshold is used to determine aggregation probability.

- *Graph reduction:* In cooperation with the aggregation algorithm, an algorithm has been also developed to reduce the number of graph links. An online graph-reduction algorithm is proposed for the deletion of transitive graph edges starting from root to leaf nodes. It is executed during the initial phase of correlation to eliminate graph complexity. A further graph reduction is performed by an offline algorithm starting from leaf to root nodes.

- *Event generation:* The ultimate goal of the proposed system is to generate security events; hence an event-generation algorithm has been presented. An event refers to the description of an attack scenario reflecting a global view of intrusion. Each event has a title and two events can be combined if they are related to the same scenario. We have also provided facilities to interact with the detected events through administrative tools.

- *Prediction of undetected intrusion:* Other approaches have dealt with broken scenarios caused by missed alerts by repairing them based on building a potentially large amount of links. However, the attack may be missed due to being a 0-day attack, where no pattern is known. An implicit mechanism has been proposed to estimate undetected activities using a generalized formalization of attack capabilities and intrusion categories. The missed attacks are not described specifically; instead a possible attack plan is predicted.

4) Knowledge modelling:

Two knowledge bases have been proposed: internal and external. We have made a distinction of abstracted attack concepts and their capabilities from dynamic information, such as vulnerability and host details. In the internal base, capabilities have been modelled using a hierarchical method based on attack classes and inheritance

between these classes. The external base represents an extendable collection containing vulnerabilities, services, OSs and host information.

5) Implementation of the proposed algorithms in a tool:

In order to evaluate the proposed algorithms in a practical manner, we have implemented these algorithms and the knowledge bases in the MARS tool. The MARS core is an engine that is capable of analysing the receipt of alerts from IDS sensors and automatically constructing security events. The attack scenario is visualised in the form of nodes and edges and the administrator is able to navigate each element for further details. The resulting attack graph is kept as simple as possible, whilst at the same time providing rich information can be obtained by request.

6) Comprehensive evaluation methodology to test the developed tool:

We have evaluated our system using a collection of different datasets. A test-bench has been set up and we have conducted a series of experiments exploiting various situations. A set of evaluation criteria has been presented including functionality, accuracy and completeness, reduction, and performance tests. We have evaluated our approach not only on the basis of the number of correlated alerts, but also using the number of correlation instances for each alert in order to achieve precise results.

1.7 Thesis outline

Chapter 2 presents background information as an introduction to the topics of the thesis, namely intrusion detection systems (IDSs) and alert correlation systems. We start with a summary of the principle concepts of IDSs, discussing models, architectures and deployment scenarios. Then, state-of-the-art alert correlation and management approaches are reviewed, including similarity-based, pre- and post-conditions based,

and probabilistic approaches. The requirements of the design and implementation of a practical alert correlation system are also discussed.

Chapter 3 lays out this study's initial research phase to carry out a performance evaluation of NIDSs. The evaluation methodologies of IDS performance have been investigated to provide a background to our preliminary testing. Extensive testing scenarios are implemented on a highly sophisticated test-bench using various platforms and configurations. A detailed performance investigation of Snort as a de facto IDS standard is given using different traffic conditions. The tests are conducted on host and virtual system configurations to explore the system response in different deployments. We also discuss packet dropping as an identified limitation of software-based IDS in high-speed environments. The chapter concludes with how the problem of missed attacks can be mitigated regardless of the reason with the use of alert correlation mechanisms.

Chapter 4 describes the core concepts proposed in this thesis: the alert correlation framework and its algorithms. The underlying *requires/provides* model with our definitions of capabilities and concepts are presented. We explain in detail the design and representation of our knowledge bases and how IDS signatures are modelled. Then, a set of proposed algorithms are described including: alert correlation, alert aggregation, event combination, event generation, and graph reduction. Therefore, issues in relation to attacks missed by the IDS have been discussed and our approach for predicting the security status.

In Chapter 5, the implementation and design specifications of the proposed framework are presented. We illustrate the MARS tool architecture, its integrated components and the system process flow.

In Chapter 6 the effectiveness of our implemented approach is demonstrated using a series of experiments. The evaluation methodology and testing criteria are discussed and the evaluation metrics are explained. We then continue to provide complete information about the datasets and experiment steps. We start with the DARPA [55] dataset evaluation for comparative purposes, incorporating the dataset description and analysis of obtained results. We then conduct two lab experiments reflecting real-life attacks to measure system functionality and performance. At the end of this chapter, a performance evaluation is presented comparing MARS and the IDS in respect to resource consumption.

Chapter 7 summarizes the thesis, reviewing our main observations and contributions. We conclude with a discussion of related research directions and promising avenues for future research.

1.8 Publications

The record in this section describes the publications presented in this thesis and group research.

Year 2009

Papers Published in Refereed Journal

- *Multi-Tier Evaluation of Network Intrusion Detection Systems*, Journal for Information Assurance and Security (JIAS), pp. 301 – 310, Dec. 2009. ISSN: 1554-1010.

Papers Published in Refereed Conferences

- *Evaluating Intrusion Detection Systems in High Speed Networks*, In Proc. of IEEE Fifth International Conference of Information Assurance and Security (IAS 2009), China, 2009, pp. 454 – 459. ISBN 978-0-7695-3744-3.
- *Snort Performance Evaluation*, In Proc. of Twenty Fifth UK Performance Engineering Workshop (UKPEW 2009), Leeds, 2009, pp. 136 – 143. ISBN 978-0-9559703-1-3.
- *Virtualization in Network Intrusion Detection Systems*, In Proc. of 4th Intl Symposium on Info Security (IS'09), Vilamoura, 2009, pp. 6 – 8. Springer Verlag ISBN 978-3-642-05289-7.

Year 2010

Papers Published in a Book

- *Virtualization Efficacy for NIDS in High Speed Environments*, In Information Security & Digital Forensics, vol. 41, Dasun Weerasinghe, Ed, Springer Verlag, 2010, pp. 26 – 41. ISSN 1867-211 ISBN 978-3-642-3-11529-5.
- *Smart Logic - Preventing Packet Drop in High Speed Network Intrusion Detection Systems*, In Information Security & Digital Forensics, vol. 41, Dasun Weerasinghe, Ed, Springer Verlag, 2010, pp. 57 - 65. ISSN 1867-211 ISBN 978-3-642-3-11529-5.

Papers Published in Refereed Conferences

- *MARS: Multi Stage Attack Recognition System*, In Proc. of the International Conference on Advanced Information Networking and Applications (AINA), Perth, 2010, pp. 753-759, ISSN: 1550-445X *(Attained Best Paper Award).*
- *High Speed NIDS using Dynamic Cluster and Comparator Logic*, In IEEE 10th International Conference on Computer and Information Technology (CIT 2010), 29 June - 01 July, 2010, Bradford, UK.

- *Detection of Coordinated Attacks using Alert Correlation Model*, In Proc. of IEEE Conference on Progress in Informatics & Computing held from 10-12 Dec, in Shanghai, China, 2010.

Year 2011

Papers Accepted in a Book Awaiting Publication

- *Implementation and Evaluation of Network Intrusion Detection System on Commodity Hardware*, Performance handbook, Next Generation Internet: Performance Evaluation & Applications, Springer, SPIN 12440030, LNCS 5233. ISBN 978-3-540-99500-5.

Papers Submitted in Refereed Journal

- *Intelligent Anomaly Detection Filter in High Speed Network Intrusion Detection Systems,* submitted in the proceedings of Elsevier Journal of System and Software Engineering.

Papers Published in Refereed Conferences

- *Event-based Correlation Systems To Detect SQLI Activities*, submitted in the International Conference on Advanced Information Networking and Applications (AINA), Bioplois, Singapore, March, 2011.

CHAPTER 2: BACKGROUND AND RELATED RESEARCH

2.1 Intrusion Detection Systems (IDSs)

The widespread use of corporate networks with sophisticated technologies, e.g. Web services, distributed databases and remote access, has raised concerns in terms of security issues. Network Intrusion Detection Systems (NIDSs) are one of the major techniques used to protect such networks against well-planned penetration. Conventionally, to secure computer systems, network services and running applications, resort was made to the creation of protective "shields". Security mechanisms such as firewalls [2], authentication mechanisms and Virtual Private Networks (VPN) have been developed in order to protect the systems of organizations. However, these security mechanisms have almost inevitable vulnerabilities and are usually insufficient in ensuring the complete security of the infrastructure. Attacks are continually being adapted to exploit the system's weaknesses, often caused by careless design and implementation flaws. This accounts for the need for security technology that can monitor systems and identify security policy violations. This is called *intrusion detection*, and complements conventional security mechanisms [56].

Understandably, intrusion is popularly defined as a malicious and externally or internally induced operational fault. Nowadays, computer intrusions and attacks are often regarded as synonymous. But more technically, an attack is an attempt to intrude (into what is supposedly a secure network), while an intrusion is actually the result of an attack that has been partially or completely successful [57]. "Intrusions in the computer systems are usually caused by attackers accessing the systems from the Internet, or by authorized users of the systems who attempt to misuse the privileges given to them and/or to gain additional privileges for which they are not authorized" [57]. Hence, the

difference that intrusion is a consequence of attack, however, unsuccessful attack is not necessary to result in an intrusion. Therefore, throughout this thesis, both terms are used from the viewpoint of the defender, and thus preventing an attack is inclusive of stopping an intrusion.

An IDS is a system for detecting and preventing such intrusions. A technical definition provided by the National Institute of Standards and Technology [58] is that it is "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer network". An IDS satisfies its reason for being by observing the network traffic or looking at OS events [59]. An IDS can be defined as "a combination of software and/or hardware components that monitors computer systems and raises an alarm when an intrusion happens" [59].

Thus, the concept of a NIDS is to observe activities among network links to detect anomalous and misuse behaviour by acquiring information from traffic and inspecting data packets in an inline or offline fashion. Then, these systems notify administrators or respond to detected threats by blocking any malicious packets or sessions. Hence, proactive systems that identify the violation of security policies are called NIDSs, whereas reactive systems that respond and stop any misuse behaviour are called Network Intrusion Prevention Systems (NIPS). However, most of these systems can be switched between the two modes based on organizational needs.

Despite both systems NIDS and NIPS perform the same analysis looking for signs of intrusion, they differ in how to provide protection for network environment. NIDS is a passive device watching the traversed packets from a monitoring port or SPAN port

(Switched Port Analyzer), matching the traffic to a set of configured rules, and triggering an alarm in case of suspicious activities. The ideal deployment of NIDS is to be connected to a monitoring port of a backbone switch as shown in Figure 2.1. A copy of network packets seen on any switch port is sent to the monitoring port to be analyzed by the NIDS. NIDS cannot block the connection and need the administrator response to deal with the detected events. NIPS have all features of the NIDS but it can block malicious traffic immediately by terminating the network connection, attacking user session, or by blocking the access to victim machines or services. Therefore, NIPS needs more tuning to keep the false positive rate to the minimum which affect the legitimate traffic. NIPS are typically deployed inline behind the firewall to limit the inspected traffic in order to improve the efficiency as shown in Figure 2.1.

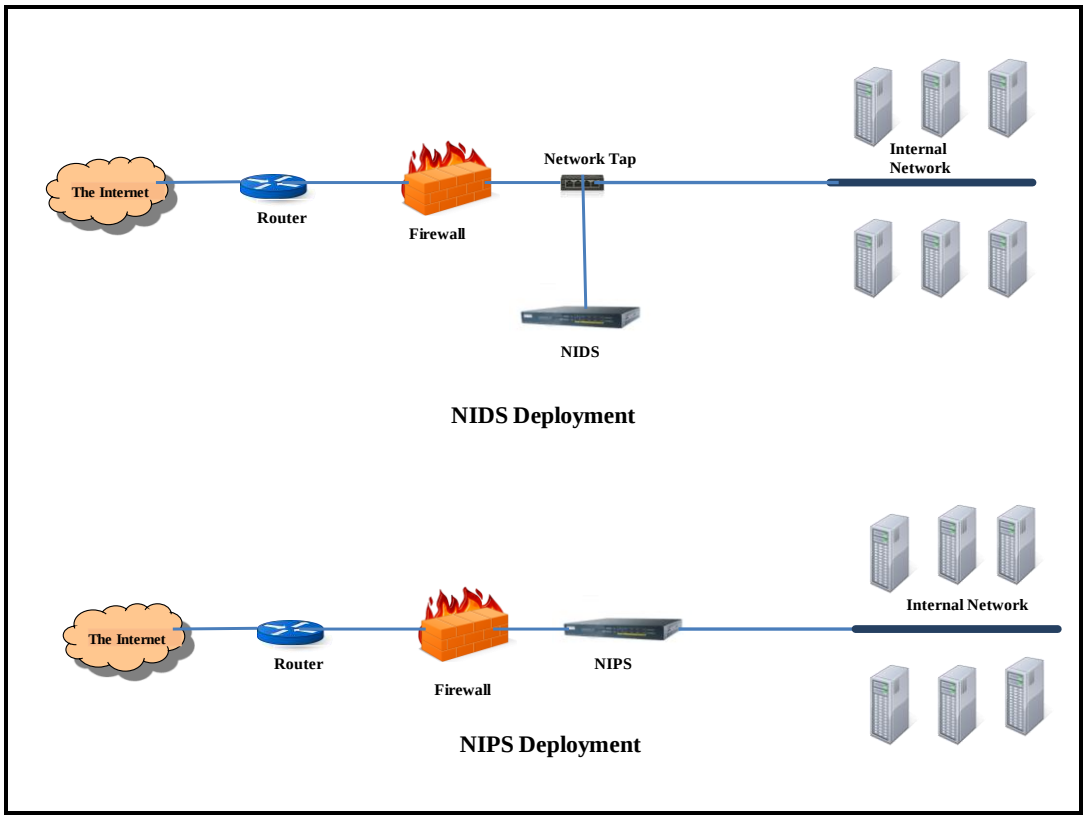

Figure 2.1 NIDS and NIPS deployment.

The notion of the IDS was first introduced in 1980 by James Anderson [60], who proposed an anomaly detection approach based on the distinction between the

characteristics of normal and anomalous behaviour. A threat model was presented that classified threats as external penetrations, internal penetrations and misfeasance. Denning [61] in 1987 introduced a general model for IDSs, which is the basis of many system prototypes have been developed since then. Denning's model includes an identification of two different models of intrusion detection systems: 1) the misuse (or signature) model, when an attack is detected based on previous knowledge of its signature; and 2) the anomaly model, when an attacker is detected based on its abnormal behaviour. This notion, based on the assumption that the normal behaviour of users and systems can be characterised, enables automatic profiling. Debar [62] proposed the first IDS taxonomy based on different criteria:

(1) Detection method: behaviour-based, knowledge-based.

(2) Behaviour on detection: passive, active.

(3) Audit source location: host log files, network packets.

(4) Usage frequency: continues monitoring, periodic analysis.

(5) Detection paradigm: state-based, transition-based.

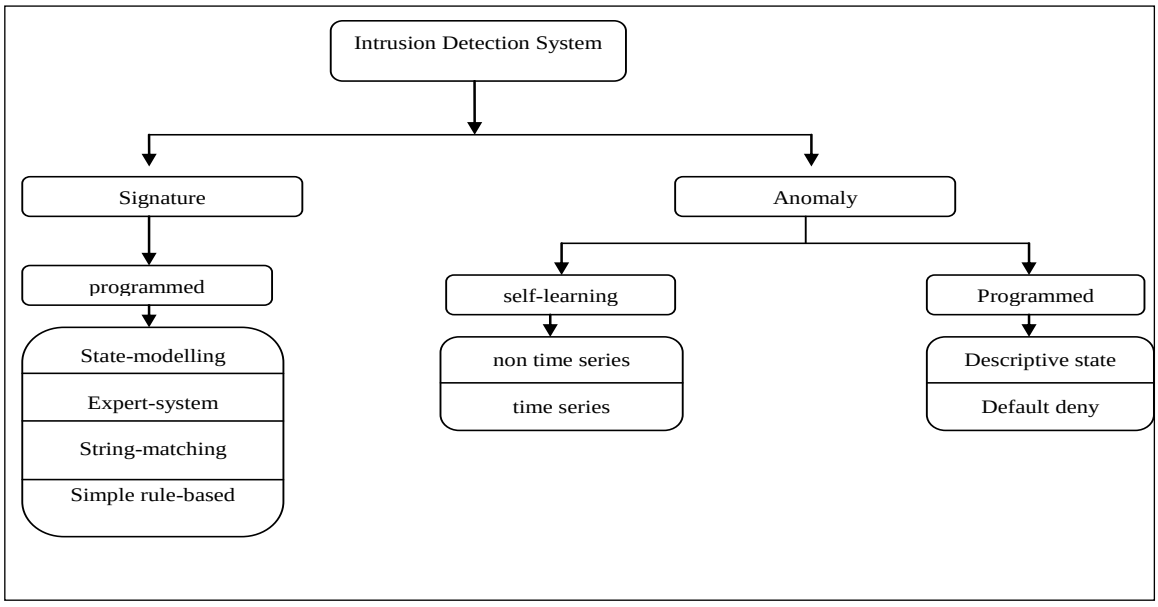

Figure 2.2 Axelsson's classification of Intrusion Detection Systems (IDSs).

Axelsson [46] proposed a generalisation model of IDSs as an alternative taxonomy, as shown in Figure 2.2. The classification is mainly based on detection principles and operational aspects

Even though several methodologies have arisen to classify IDSs since 1980, these fall into three general approaches: 1) anomaly- (behaviour) based, 2) signature- (knowledge) based, and 3) hybrid systems (anomaly and signature).

2.2 Intrusion Detection Systems: methodologies

2.2.1 Anomaly-based detection

Anomaly-based detection methods are based on a deviation of abnormal activities from the normal or expected behaviour of the system. A set of characteristics of the system are observed and analyzed to create a model of normal behaviour using collections of information about the system over a particular time interval. IDSs can detect anomalies when they compare current behaviour to the normal system model in order to identify, report and block any violation. Moreover, anomaly-based methodologies are based on the assumption that any anomaly is an indication of a potential attack.

Normal behaviour is learned by the system during an online/offline training phase (heuristic systems). Collected data from the learning stage is analysed, pre-processed and processed; then the normal model is built according to these observations. Therefore, audit data is inspected for any abnormal patterns deviating from the normal model baseline, and these are considered malicious. The effectiveness of these methodologies depends on the selected variables and parameters to build the model of the system profile [63]. These parameters vary from simple statistical data to comprehensive measures. Therefore, robustness of these systems is proportional to the amount and accuracy of measured data. In addition, these sorts of systems should be

adaptable due to the complex and changing nature of protected environments, such as communication networks. [64] summarizes the anomaly-based IDS process into three stages: 1) a *parameterization* stage to collect the observed instances of normal system behaviour; 2) a *training stage* to characterize the normal and abnormal models, which can be achieved either manually or automatically; and 3) a *detection stage* to detect any deviation exceeding a pre-defined threshold. These systems are theoretically able to detect novel and 0-day attacks [65]. However, their efficiency is strongly dependent on model construction and threshold selection. Several techniques are used to build anomaly-based systems.

2.2.1.1 Statistical techniques

The objective of statistical techniques is to observe the system's activities in order to determine its behaviour, and then to generate system profiles. Selected variables are sampled over a specific period of time to measure the normal behaviour of the system. The observed activities can be system logs, spatial and temporal characteristics of network traffic, or system calls. Two models are built: a model stored or programmed and a current model; and detection is based on the degree of abnormality in the comparison of the two models considering a threshold metric. The advantage of these approaches is that they do not require prior knowledge of the observed systems. However, one of the biggest drawbacks of these techniques is determining the threshold in order to achieve a balance between false positives and false negatives, which is difficult in the presence of different situations and requirements. In addition, intruders can deceive the protection system to send malicious data by training the target system itself.

Haystack's prototype [66] was developed as one of the earliest statistical anomaly-based IDSs. The detection system considers a combination of two models: user behaviour and

generic group behaviour. It takes into account a range of normal behaviour events between two limits and each event has a score, with a high score indicating an anomaly. However, normal system features are extracted offline only. The early proposed techniques in this respect were based on *univariate* models such as [61]; however, the trend lately has become toward *multivariate* models that consider more than one single variables [67]. Using a combination of metrics rather than only one provides more accurate discrimination between the observed models.

2.2.1.2 Expert systems

Expert systems **[**68**,** 69**]**are knowledge-based and used to build the profile of a system or its users based on rules obtained from statistical measures of normal behaviour over a period of time. Primarily, these approaches are intended for data classification according to the extracted rules. In the first stage, training data is used to define certain variables and classes, and then classification rules are inferred and applied to audit data. The W&S (Wisdom & Sense) [70] expert system was proposed to detect anomalies in user behaviour. The IDES (Intrusion Detection Expert System) [69], developed at the Stanford Research Institute (SRI), is a system that summarises user behaviour and calculates interrelated statistics, and then compares the current activities with the user profiles. The next generation of NIDES (Next-generation IDES) was designed to operate online to monitor system activities. The SPADE (Statistical Packet Anomaly Detection Engine) [71] is a Snort pre-processor plug-in, developed to use the concept of anomaly score to detect stealthy port scans. It consists of two sub-systems: an anomaly monitoring sensor and a correlation engine. An alarm is triggered if the assigned score of each event exceeds a specific threshold. The main advantage of these approaches is flexibility and accuracy; however, constructing the required knowledge is not an easy task and is a time-consuming process.

2.2.1.3 Machine learning

Learning is a process to learn the dependency between two sets of information to generate an unknown input-output model based on a limited number of observations [72]. An accurate observation that describes the constructed model requires an accurate problem definition. Machine learning techniques have been used widely in computer systems to provide intelligence in the automatic process. The tasks of machine learning include: classification, acting and planning, and interpretation. IDSs can be identified as a classification problem (with two classes: normal and abnormal) [72]. Training data captured from the normal usage of the system are used to build the model and then classify behaviours as either normal or anomalous. These systems are either generative (profiling) to learn the normal behaviour and to detect intrusion deviating from the learned profile, or discriminative to learn the distinction between normal and abnormal activities [72].

Generally, learning methods can be classified into two broad categories: supervised and unsupervised learning systems. In supervised learning, training data (labelled data) is used to generate normal and abnormal behaviour. Each training pattern is weighted to construct a detection model and the weights are adaptive to obtain a feasible and accurate system. It is required to predict model behaviour variables for any input variables after the training phase. Formally, given variables (x,y), $x \epsilon X$, $y \epsilon Y$, the objective is to find a function $f : X \rightarrow Y$ which represents the intrusion detection model. The degree of mismatch between X and Y represents the cost function of the prediction algorithm.

On the other hand, in unsupervised learning (unlabeled data approach), anomalous data is not needed; instead, a normal model is constructed from normal system patterns. For anomaly detection systems, unsupervised methods are more effective for building the

model by observation without any prior knowledge of intrusive behaviour. However, machine learning is not limited to these approaches; semi-supervised learning, active learning and deep learning are widely used in researches. Examples of machine learning systems are Y-means, neural networks and support vector machines (SVMs) [73].

Machine learning techniques for system calls analyses have been used for host-based IDSs to learn program behaviour so as to detect irregularity. Forrest et al. [74] discovered that sequences of system calls were very consistent and a normal model could be built and used to detect abnormal activities. Their work was based on *similarity function* to compare the human immune system and IDSs. [75] proposed multiple detection models for the system calls to be evaluated from different points of view. Weighted scores for events are accumulated to construct the detection model.

Bayesian methodology has been conducted by several researchers due to its unique features. It is based on probabilistic relationships among events to find or predict the cause of actions by moving back in time. [76-78] used a Bayesian network to create models for anomaly detection. In addition, Principle Component Analysis (PCA) is a technique used to reduce massive and multi-dimensional datasets to lower dimensions for analysis. Large and complex datasets are difficult to understand and process. A large number of correlated variables are transformed to a smaller number of uncorrelated variables. [79] proposed an anomaly-based detection system using PCA to reduce the audit data. [79-81] present a model that is suitable for high-speed processing, where the dataset is collected from system calls, shell commands and network traffic. Markov models are also used to detect anomalies based on sequence of events, where the system is examined at some particular time. [82] developed an anomaly detection system for systems calls based on Markov models. A hidden Markov model is also employed in anomaly-based systems where the system is assumed to be a Markov process but with

hidden parameters. [83, 84] used a hidden Markov chain to develop host-based detection systems. [85] developed several methods for network-based anomaly detection systems. In practice, these techniques generate flexible and adjustable systems, as they discover the interrelations between system variables. However, they rely on assumptions drawn about the observed system and require training data.

2.2.1.4 Data-mining techniques

Data-mining techniques have also been employed in anomaly detection systems in many researches to extract a knowledge model from a large number of patterns. Association rules from the system patterns are utilized to create features that construct the detection system. Two types of methods applied in data mining are 1) predictive methods involving certain variables to predict unknown variables; and 2) descriptive methods where human interpretation are used to detect unknown patterns. Data-mining approaches are generally applied to three main tasks: classification, clustering and association. Classification is intended to extract class attributes from training data and learn the model using the training data, and then to use the constructed model to detect the anomalous events. An example of classification techniques are: inductive rule generation techniques, fuzzy logic, genetic algorithms and neural networks. RIPPER [86] used inductive rule generation techniques to induce rules from data to classify audit data and detect intrusions. Dickerson et al. [87] developed the Fuzzy Intrusion Recognition Engine (FIRE) to derive rules for every observed event. Other approaches [88, 89] have used genetic algorithms to extract classification rules.

In the clustering and outlier detection task, patterns in unlabeled multi-dimensional datasets and the number of dimensions equal to the number of attributes are identified. [90, 91] presented outlier detection techniques to create clusters and apply rules on audit data. The MINDS (Minnesota Intrusion Detection System) [92] is considered a

clustering-based anomaly detection system. Association rule discovery mechanisms are used to correlate events usually occurring at the same time. ADAM (Audit Data Analysis and Mining) [93] is an association rule and classification based anomaly detection.

2.2.2 Signature-based detection

Signature-based detection methods are knowledge-based techniques where well-defined attack patterns are used to detect malicious security violations. A novel attack has to be studied and analysed to identify its features and then generate its accurate signatures. The detection system observes and analyses activities amongst audit data, and the detection mechanism is based on the comparison between attack signatures and observed patterns. Signatures can be defined as a set of conditions characterizing the direct manifestation of intrusion activities in terms of system calls and network data [94], which is to say that when these conditions are met, a type of intrusion event is indicated. In networks, unauthorized behaviour is detected by sniffing packets and using the sniffed packets for analysis [95].

This intrusive model is more accurate than the normal behaviour model and it does not need to observe the system's normal behaviours. In addition, it can be efficiently applied in heterogonous environments, while its detection process works independently from the normal system behaviour. The detection mechanism is based on a pattern-matching process performed on audit events.

In these systems, the collection of signatures describing malicious activities is stored in a database similar to an anti-virus system. The observed events extracted from captured data, such as network traffic packets, are compared with the pattern database, and then an alarm is triggered in case of matching. The database has to be up-to-date and the

signatures have to be accurately defined to achieve an acceptable balance between false positives and false negatives. If the signature descriptions are very specific, this will result in false negatives and missed attacks. In contrast, if the signature descriptions are very general, a large number of false positives will be generated. Snort [23, 24] is the de facto standard for IDSs, which is categorized as a signature-based detection and prevention system. However, it employs protocol anomaly inspection as well as many commercial and open-source detection systems using Snort signatures. Snort will be explained in detail later in this chapter.

Typically, these types of systems consist of two sub-systems: a sensor to collect data from its sources, and an engine to perform pattern matching. However, in an ideal scenario, signature systems are incorporated with a pre-processing mechanism such as protocol analysis to remove ambiguities from the collected data. The most expensive process in such systems is the pattern-matching process, particularly in high-speed environments. For this reason, many algorithms have been proposed in the research community to enhance the functionality of the pattern matcher [96-98].

Software-based pattern matcher systems have been used for several years, but with the evolution of Gig networks these systems have become bottlenecks. Therefore other areas of research and certain commercial products are implementing hardware-based pattern-matcher systems to utilize their high-speed processing [17, 45, 99]. Generally, the most well-known algorithms for pattern matching are Boyer-Moore [100] for single pattern matching and Aho-Corasick [101] for multiple pattern matching. For hardware-based solutions, FPGA (Field Programmable Array Gates) and TCAM (Ternary Content Addressable Memory) are implemented for their parallelism capabilities [102]. [44] found that 87% of Snort rules have patterns, so he proposed a hardware accelerator for

pattern matching. [99] proposed a high packet processing system using TCAM. An Extended TCAM was proposed by [103] to reduce data structures.

Certain efforts have dealt with software-based solutions to enhance performance. [104] proposed a method to process each packet once it arrives without reassembly and to integrate pattern matching in protocol analysis to reduce execution time and memory use. [105] integrated pattern matching, normalization and protocol analysis in pro-to-matching techniques to improve Snort functionality.

Each signature-based and anomaly-based IDS has its advantages and disadvantages. The signature-based IDS is more practical and widely deployed because the intrusive model is easier to develop to meet security policies in heterogeneous environments. More precise definitions of signatures lead to more precise detection and reduced potential of missing attacks (false negatives). Comparatively, false positives in such systems are considered lower than in anomaly-based systems because the detection mechanism is based on matching patterns of activities to knowledge of attack patterns. In addition, alarms generated by these systems provide the administrator with detailed and precise information about the intrusion and the attack actions. On the other hand, signature-based systems cannot recognise 0-day attacks due to the absence of corresponding signature definitions. The system can also be evaded by altering signature patterns in a way that does not affect the ultimate goal of the attack, such as mutant exploits or polymorphic behaviour (self-modifying behaviour). Keeping up-to-date with new vulnerabilities along with the maintenance burden are further drawbacks of these systems.

In contrast, the anomaly-based system has the ability of detecting novel attacks without prior knowledge and without the need to create new signatures for each unforeseen

exploit. This can be efficient in the detection of Internet worms and similar stealthy attacks. Vulnerability updates are not required as a result of considering any suspicious activity as potentially malicious. On the other hand, anomaly-based systems suffer from difficulties in precisely characterising normal behaviour models in order to create baselines of detection. Determining the degree of deviation from the norm to provide reasonable detection accuracy is another obstacle. Moreover, these types of systems require a training phase including intensive analysis of the target environment. And any development fault in this phase can cause the generation of a large number of false positives. Furthermore, modern methodologies of attack tend to be slow-and-low, without creating a noticeable deviation from the normal model of the typical system, thus such malicious activities cannot be detected. Moreover certain emerging attacks, such as cross-site scripting (XSS) [106] and code injection, are categorised under the normal usage of any system, which makes them difficult to detect. The changing nature of network systems (burst networks) may result in high false alarms, even though the normal behaviour is well defined. Finally, the generated alarm reacting to abnormal activity does not give specific information to the administrator about the attack.

In fact, neither of the two is the panacea. When used in conjunction with each other, then each of the two become a more viable and effective means of protecting network infrastructures [94]. The signature-based IDS still serves as a good outer layer of defence against known attacks in the same manner as firewalls. Anomaly-based IDSs are employed to further fortify the defence system and do not serve to function in lieu of signature-based IDSs [107]. Today, it is being observed that numerous antivirus packages include both signature-based and anomaly-based detection features, while only a handful of IDSs effect an incorporation of both approaches.

2.3 Hybrid IDSs

The recent trend in the intrusion detection research community is to have the above approaches to interoperate efficiently and manipulate their positive features so as to achieve maximum levels of protection. Signature-based systems provide accuracy and less false positives, and anomaly-based systems offer recognition of novel attacks. Figure 2.3 shows the typical architecture of hybrid systems, where a signature-based sub-system such as Snort receives the incoming network data and performs monitoring using a protocol analysis unit and a pattern matching unit. If a malicious activity is detected, an alarm is triggered and there is no need to pass the captured data to the anomaly sub-system. Otherwise, the data is transferred to the anomaly sub-system for further observation. Therefore, only traffic supposed to be benign is forwarded to the receiving applications, and malicious activity is detected. Then the detected suspicious behaviour is further analysed by experts, and potentially a corresponding signature can be generated for future use.

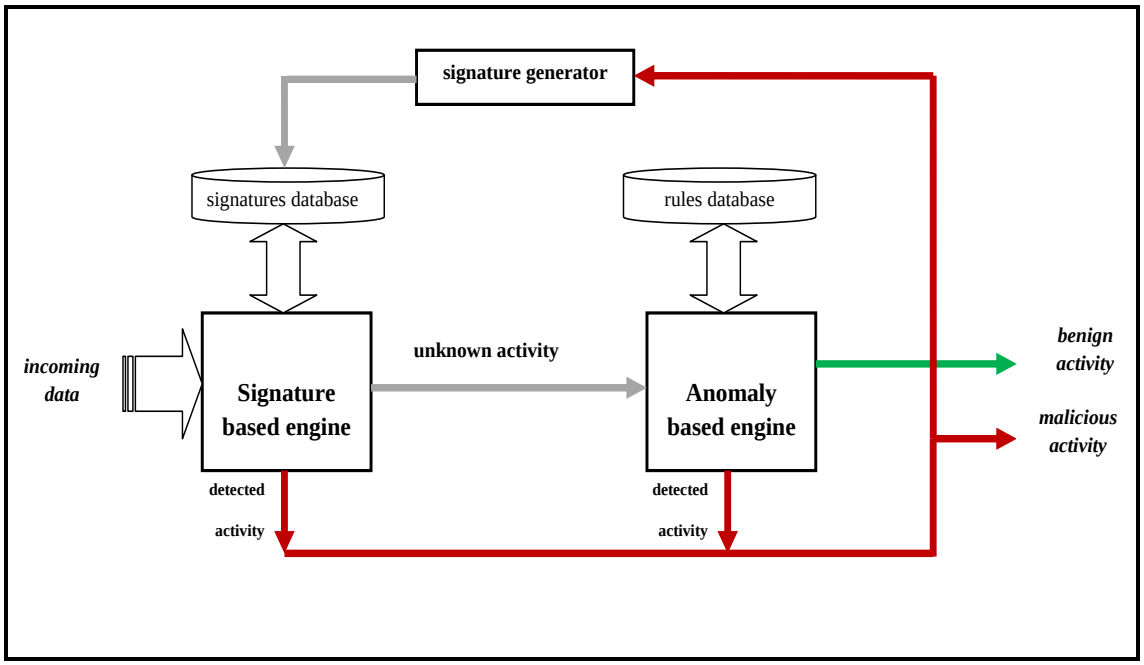

Figure 2.3 Architecture of hybrid systems.

2.4 Snort

Open-source software has gained tremendous popularity and acceptance amongst academia and the research community. Apart from being free of costs, there are several other qualities that have made them popular. Some of the advantages of open-source software are access to source code, detailed documentation, online forum support and rights to modify/use. Our research has focused on a widely accepted open-source software tool, Snort [20]. Snort has received great acceptance in the IDS market and has been widely recognized as the reliable open-source tool.

Snort is capable of performing real-time traffic analyses and packet logging on the network. It performs protocol analysis and can detect a variety of network threats by using content/signature matching algorithms. Snort can be configured as a packet sniffer, packet logger and NIDS (detection mode and inline mode).

- *Sniffer mode*: To receive traffic packets from the traffic wire and display them exactly the same as function of TCP dump. Snort uses a libpcap library for packet acquisition.
- *Packet logger*: This is similar to the above, in addition to storing the data on a disk.
- *Network intrusion detection*: The main task for Snort to perform is traffic analysis and pattern matching against signature collections.
- *Inline mode*: (or network intrusion protection mode): To acquire traffic packets from *iptables* instead of *libpcap*. Attacking packets according to Snort rules are dropped instantly and only benign traffic will be forwarded.

Snort was introduced in 1998 by Marty Roesch [20], and was considered a signature-based IDS. Since its early versions launched in 1999, many development efforts have

been implemented to improve its capabilities. The current version is 2.8.6, and more than 8,000 certified rules are included. SnortSP 3.0 [23] is the beta version with new architecture introducing a new shell-based user interface. The Snort system consists of four sub-systems working sequentially, as shown in Figure 2.4.

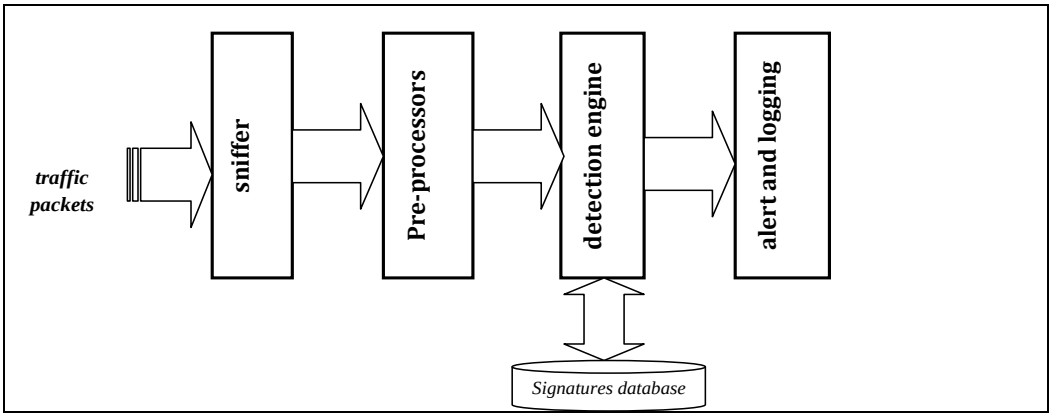

Figure 2.4 Snort sub-systems.

Snort has five components: 1) a packet decoder, 2) a pre-processor, 3) a detection engine, 4) a logging and alerting system, and 5) an output model. Incoming packets are prepared for processing before being modified if required, e.g. de-fragmentation before sessions are then reassembled. Snort rules are applied in detection engines, where they are examined against signatures to detect recognised attack patterns.

2.4.1 Pre-processor

Pre-processors have been introduced to run before detection engines to improve Snort protection speed and efficiency. They are intended to perform traffic normalization to detect protocol anomaly behaviour. They are based on a target-based technique inspired from Patcek and Newsham's paper on evasion of attacks [11], and Vern Paxon and Umesh Shankar's paper [12] on traffic normalization. The heterogeneous nature of communication network infrastructures has posed ambiguities due to various interpretations of the RFCs [108]. The target-based analysis [23] involves identifying the actual target characterisation in order to provide the IDS with additional information

about the protected network so as to defend against attack evasions. Different OSs can behave in different ways in terms of handling network traffic, and the IDS must understand how these OSs are functioning. Intruders may manipulate these ambiguities in protocol implementations by fragmentation and session-splicing techniques. Pre-processors in Snort consist of:

- Packet de-fragmentation to reassemble traffic data spread over multiple packets.
- Session reassembly to provide a stateful TCP analysis by using state records of previous TCP connections.
- An application pre-processor to normalize ambiguities in application-level protocols, such as Telnet, HTTP, SMTP, FTP and RPC protocols.

Dynamic pre-processors are plug-in pre-processors dynamically loaded and separately developed, and compiled without the need for full Snort compilation.

2.4.2 Detection engine

The main task of a detection engine is to perform the pattern-matching task. It receives the data from pre-processors and matches the packet header and content against Snort signature rules. Snort, being a signature-based IDS, uses rules to check for hostile packets in the network. Rules are sets of requirements used to generate an alert and have a particular syntax. For example, one rule that checks for peer-to-peer file sharing services looks for the string “GET” in connection with the service running on any port other than TCP port 80. If a packet matches the rule, an alert is generated. Once an alert is triggered, it can be sent to multiple places, such as a log file or a database, or it generates a Simple Network Management Protocol (SNMP) trap [109]. On successful detection of a hostile attempt, the detection engine sends an alert to a log file through a network connection into the required storage (output) [24]. Snort can also be used as an

Intrusion Prevention System (IPS) [24]. Snort 2.3.0 RC1 integrated this facility via Snort-inline into the official Snort project [23].

The main objective of Snort and other NIDSs is to effectively analyze all packets passing through the network without any loss. The performance of the majority of running applications depends upon memory and processing power. In the context of NIDSs, this performance dependency includes NIC cards, I/O disk speed, and OS. In recent years, technologies have advanced in both hardware and software domains. Multi-core systems have been introduced to offer powerful processing functionality. However, these multi-processing implementations support applications using concurrent programming. The number of CPU cycles in such systems has increased to execute multiple tasks simultaneously.

It has been identified that Snort does not support multithreading [24]. The detection engine component of Snort constitutes the critical part where the pattern matching function is performed. Recent VRT rule libraries contain more than 8,000 rules; this augments the need for an effective pattern matcher. Snort uses three different pattern matching algorithms: Aho-Corasick [101], modified Wu-Manber [110], and low-memory key-word tire [24, 96]. Modifications have been made for these algorithms to provide various performance characteristics. We have conducted comparative memory usage and performance tests for different pattern-matching algorithms. The results are shown in Table 2.1.

Table 2.1 Pattern-matching algorithm performance (based on 1.5 GB pcap file).

Algorithms (8,296 rules)	**Memory usage (MB)**	**Packet processing time (seconds)**
Aho-Corasick (full)	640	620
Aho-Corasick (sparse)	240	714
Aho-Corasick (standard)	1,080	665
Wu-Manber	130	635
Wu-Manber (low)	75	655

In addition, Snort uses Perl Computable Regular Expressions (PCRE) [24] for precise and flexible protection capabilities. A dynamic engine is also used for complex detection functionalities where shared objects are dynamically loaded. Instead of plaintext rules, rules can be written in C language and compiled and loaded for fast processing and to deal with certain complicated attack vectors. Snort rules consist of rule headers and rule options. The structure of rule headers is shown in Figure 2.5 and multiple rule options are enclosed in parentheses. An example of a Snort rule is provided below:

```
alert tcp 192.168.2.0/24 23 -> any any \
(content: "confidential"; msg: "Detected confidential";)
```

Action	Protocol	Src. Address	Port	Direction	Dest. Address	Port

Figure 2.5 Snort rule header.

2.4.3 Snort with Artificial Intelligence (SnortAI)

SnortAI [111] has been introduced to integrate Snort methodology with the intelligence of anomaly-detection methods represented by Artificial Intelligence plug-ins. Currently, portscan-AI pre-processors function with Snort version 2.8.3.2 and the development of other plug-ins, such as XSS-AI and SQL-AI pre-processors, is in the planning.

2.5 Bro

Bro [21] is also an open-source IDS to parse network traffic in real-time focusing on extracting application-level semantics and event observations. It was developed at ICST and LBNL [112] in 1996. The detection of a specific attack is implemented by comparing activities against a set of rules and policies. However, Bro does not look only at pre-defined signatures, but analyses network connections and correlates between events. Moreover, it uses regular expression matching and a DFA (Deterministic Finite Automaton) [113], where one active state is used at a time.

Bro system architecture consists of three main sub-systems: 1) a sniffer to capture traffic, 2) an event engine, and 3) a policy script interpreter. When network packets are captured, different levels of events are generated by the event engine (the core). Streams of produced events are transferred to the policy script interpreter to be processed. Polices are either supplied by the administrator or acquired from the connection context analysis. Events are handled by the event handler following rules specified by policy scripts. Policy scripts have to be written in Bro script language. However, Bro suffers from a shortage of good documentation, slow development and the need for writing complex scripts.

2.6 Host-based vs. network-based IDSs

IDSs can be categorised based on the source of gathered information for observation and analysis. Host-based IDSs were introduced before network-based IDSs to monitor the activities on a single host. These activities include file access and modifications and the detection is achieved by checking file integrity, kernel activities such as system calls, and root privilege behaviour. Furthermore, connection attempts can be observed such as suspicious port connections and failed logon attempts, as well as application-level interaction. Records of information are collected and analysed against any intrusion attempt. Examples of such systems are: tripwire [114] – a software for security and data integrity, and OSSEC[115] – an open-source host-based IDS.

In contrast, network-based IDSs (NIDSs) monitor local network activities by analyzing inbound and outbound traffic in real-time. All traffic packets – captured from network interface on a promiscuous mode – are reassembled and analyzed using different mechanisms. Certain network-based attacks, such as distributed denial of service (DDoS), Botnet and worms, cannot be detected by host-based IDSs. Thus, NIDSs are

efficient because they protect the network elements, including host machines, largely without having to rely on frequent OS patches and user awareness. They also reduce the cumbersome task of installing and updating protection software on every single host. The main concerns about such systems are when they become a bottleneck for network communications, particularly with the massive speed of modern switches. The other concern is the difference in understanding of the received data between the NIDS and the end application. Also, application-level attacks, which need application-layer inspections, have posed another challenge for such systems. The research trends in this area are to enhance NIDSs to be more intelligent in understanding attacker behaviour. This is achieved by incorporating certain functionalities implemented in host-based tools.

2.7 Alert correlation

The widespread deployment of IDSs, both network-based and host-based, has imposed a demand for sophisticated alert management systems. Simple analyses including statistical information about IDS alarms are not helpful in the detection of connections between alerts, in reducing alarm-data size and in distinguishing false alarms. A high-level view of system security status is required by analysing low-level alerts produced by IDSs to characterize attack actions. Alert correlation techniques provide the facility to observe beyond the receipt of IDS alarms themselves. It has been identified that real intrusion consists of multiple and coordinated steps that are logically connected. In addition, the huge amount of elementary alerts received constantly can cause the human administrator to ignore them if they consist mostly of false positives.

Alert correlation has been an active research area for many years and the concept has been explored in several efforts. Approaches have been developed for Network

Monitoring Systems (NMSs) to diagnose faults in complex communication networks, and have also been applied to some extent in alert correlation. However, the nature of network faults is different from adversarial behaviour, as the later is more dynamic and complex. In NMSs, the objective is to find out the fault location, whereas in the correlation of IDS alerts, the goal is to discover the attacking strategy.

In recent years, alert clustering and correlation techniques have been employed to provide a global view of attacking behaviour by analyzing low-level alerts produced by the IDS sensors. The main objective of alert correlation is to build an abstract modelling of alerts by generalizing the detected events, instead of the current specific modelling. The constructed inference will progress even in cases of unforeseen attacks. Previous research efforts in the field of alert correlation have mainly concentrated on a particular aspect of the problem domain. It is not possible to provide an efficient alert correlation system in a single phase or study the system components as isolated elements. Overall functionality is only achieved by the integration of the system's modules and all the system parts should be evaluated together.

Different approaches have been utilized to build the correlation models and can be categorized into four main disciplines: 1) similarity-based approaches, 2) scenario-based approaches, 3) pre- and post-condition approaches, and 4) probabilistic approaches.

2.7.1 Similarity-based approaches

In similarity-based techniques, certain selected features (e.g. source IP address, destination IP address, port number and attack class) are used to compute the similarity degree. Some approaches rely on exact similarity between two alerts to be grouped, e.g. the same source and destination IP address, whilst others utilize a similarity function. This function represents similarity confidence and is based on a defined threshold. Two

alerts are considered similar if they satisfy the defined confidence degree and occur within a defined sliding window time. The similarity confidence is calculated using the overall similarity between alerts based on their attributes. In principle, these techniques are mainly applied to *alert fusion, alert clustering* and *alert aggregation.* The Alert clustering process plays an important role in alarm reduction and as well as reducing false positive rates. Data mining, artificial intelligence, machine learning and clustering using association-rules techniques are widely implemented in this respect.

[50] has proposed an algorithm for alert aggregation and correlation which is implemented in the Tivoli Enterprise Console (TEC). It is a tool for risk management to address the problems of alarm flooding and discovery of attack context. It has two different components: one to remove duplicated instances of alerts using rules saved in a configuration file, and the other to assign alerts to their associated attack scenario. An exact similarity of three common attributes (attack class, source address and destination address) are used to group alerts. It is useful for some initial alert processing but is not capable of detecting complex scenarios. It is also vulnerable for attack flooding and the use of different IP addresses for the same attack. However, their alert model has been revised and is now the de facto standard format for intrusion detection alerts, which is the Intrusion Detection Message Exchange Format (IDMEF) [116]. An alert correlation framework was presented by [51] using exact feature similarity. Two out of ten of the proposed components are implemented. *Thread reconstruction* is used to cluster alerts with equal source and destination IP addresses within a window size. This is to represent the activity of attacking a single host from a single attacker. Another component called *focus reconstruction* is used to show a single attacker targeting multiple machines.

In other respects, approximate features similarity is used by [28], who presented a probabilistic approach to provide a unified mathematical framework that performs a partial matching of features. Features are extracted and minimum similarities are computed and weighted. A similarity metric is employed using EMERLAND architecture [28] in three phases. In the first phase, an attack thread concept and similarity metric (sensor, attack class, source and destination) are used to aggregate low-level events. The second phase involves the aggregation of alerts generated from multiple sensors ignoring sensor field information. Then the third phase provides a higher aggregation level by relaxing the similarity requirements using attack class.

Although these methods are useful for alert fusion and statistical purposes, they fail to discover the causal connections between alerts. Moreover, it is hard to find a justification for calculating the overall similarity function using a weighted measure and sliding window time.

Additionally, a conceptual alarm clustering technique was proposed by [10] to discover root causes of different alarms. The aim was to reduce the volume of the alarms to a manageable size. A generalization hierarchy structure of attributes was utilized to define similarities between alerts and to support root cause analyses. The similarity function is computed using the proximity between alerts and the features' taxonomy [10]. In essence, the generalisation concept is promising, but not in certain features, such as IP addresses if spoofed ones are considered. The concept of generalisation has been utilized but for attack class classification and capabilities modelling.

2.7.2 Scenario-based approaches

Scenario-based or pre-defined scenario approaches utilize the concept of the real attack consisting of a series of steps to achieve the attacker's ultimate goal. Attacks occur

typically in groups of actions (multi-stage attacks) represented by IDS alerts. Each attack scenario is specified by its corresponding steps, which are required for it to be successful. Attack scenario modelling is essentially based on rules stored in a knowledge base that states attack stages. The knowledge rules are built either manually by experts or using machine learning approaches. The knowledge base is generally intended to characterize the casual relationships between observed attack activities. In manual knowledge acquisition, formal detection models using attack languages [33, 117, 118] are used to construct attack libraries. On the other hand, in machine learning approaches, correlation rules can be obtained using a training stage and labelled data.

LAMBDA [33] is an intrusion specification language to describe the conditions and effects of an intrusion in connection to the variable state of the target system. Descriptions of the relationships between attack steps are constructed based on three components. The first component is termed the *state description*, which is to specify the conditions of the target system that are required for the attack to be successful. The other components are termed *transition description* and *event combining*, to state the conditions in order to combine two events into a single scenario. ADele [118] was presented at the same time of development as LAMBDA. However, it is a procedural approach rather than the declarative mechanisms utilized in LAMBDA. A database of known attack scenarios is modelled in a high-level description. Similarly, STATL [117] language is a formal language to describe scenario patterns in terms of states and transitions. Hence, a sequence of events conducted by the attacker can be described to express a multi-stage attack. However, these approaches need a manual description of potential attacker behaviour, and if a single step is missed the whole behaviour goes undetected.

Besides, [119] used predictive data-mining techniques to learn correlation algorithms from labelled scenarios. The training data is obtained from real scenario examples and labelled manually. A user-defined threshold is used to determine the highest probability score, stating whether the incoming alert corresponds to a particular scenario or otherwise to initialize a new one. [120] applied chronicle formalism to alert correlation to provide fewer alarms of higher quality. The proposed approach is based on known sequences of malicious scenarios and temporal logic formalism. The chronicle model incorporates a formal data model M2D2 [120], which is also proposed by the authors to federate the context information required for alert correlation systems. In practice, these approaches involve data labelling, which is labour intensive and error prone. Furthermore, the training data which can be relied on and which reflects real scenarios is not available.

2.7.3 Pre- and post-condition approaches

The basis for these approaches is the assumption that real attacks involving related stages can be represented by alerts as a system diagnoses. The objective is recognition of attack scenarios, and potentially the identification of unknown attack steps. Domain knowledge of intrusion pre- and post-conditions is used to detect alert correlation even with the existence of partial condition formalisation. Two alerts can be logically correlated if some of the post-conditions for the first one match some of the pre-conditions of the later one. It can be said that these techniques are a special case of scenario-based approaches; however, complete scenario description is not required. Hence, if some steps are missing due to not be detected by the IDS, the correlation system can perform a correlation to detect the so-called attack sub-goal. This provides more tolerant techniques than the hard-coded scenario templates used in scenario-based methods.

The *provides/requires* model was initially proposed by [121] to characterize the causal relationships among alerts using JIGSAW language [121]. Attack scenarios are modelled in terms of *capabilities* and *concepts.* Concepts are abstractions of attacks, and capabilities are the required and provided conditions associated with each attack concept. The correlation task is performed if a match is detected between the conditions of two alerts ordered temporally. Hence, each received alert is modelled to a concept with its related required and provided capabilities. Instead of representing attack scenarios as series of states, they are considered as sets of concepts and capabilities. Even though it is limited to known attacks, the formalization of concepts and capabilities can be generalized in a hierarchical manner to uncover unknown atomic activity. Several efforts have been proposed based on this model in the literature, but they have used various definitions and knowledge representations [35, 38, 49]. We have used this model as the basis of our correlation framework because of its extensibility and flexibility.

[49] proposed the Cooperative Intrusion Detection (CID) framework based on pre- and post-conditions. Explicit correlation of events based on security experts is used to express the logical or topological links between events. The framework consists of five components: 1) alert management, 2) clustering, 3) merging, 4) correlation, and 5) intent recognition. Alert clustering and merging functions are performed using a similarity function, and intent recognition is not implemented. The attack is specified in the language of LAMBDA [33] and partial matching techniques are adopted to construct attack scenarios. In addition to explicit correlation, semi-explicit correlation is used to overcome the possibly missing attack descriptions. Moreover, the authors of [34, 35] have proposed an alert correlation framework based on the prerequisites and consequences of individual detected alerts. A Hyper-alert Type Dictionary knowledge

database contains rules that describe the conditions where prior actions prepare for later ones. The attack strategy is represented as a Directed Attack Graph (DAG) with constraints on the attack attributes considering the temporal order of the occurring alerts. The nodes of the DAG represent attacks and the edges represent causal and temporal relations. Similarities between these strategies are measured to reduce the redundancy. A technique of hypothesizing and reasoning about missed attacks by IDSs is presented to repair broken scenarios. This is done by matching instances of prerequisites and consequences of similar attack nodes. The main objective of these authors' work is the reduction of the huge number of redundant alerts and to report a high-level view for the administrator. However, the proposed system is useful as a forensic tool where it performs offline analysis. In addition, building the knowledge database containing rules of the applied conditions is burdensome. However, the authors have not provided a mechanism to build the Hyper-alert Type Dictionary. Moreover, the generated graph is huge, even with medium-sized datasets.

In spite of the fact that pre- and post-condition approaches have alleviated some of the recognised drawbacks of scenario-based approaches, they also share the difficulty of defining the required knowledge. Pre- and post-conditions have to be modelled for every known attack and this is typically done manually by security experts. The quality of correlation results is highly dependent on how attack elements, attack implications, attack domains and the target system response are expressed. Attack concepts have to be formalized in a certain way to provide maximum coverage with less false positives. Furthermore, some implementations of these techniques consider uncorrelated alerts as false positives, and that is not the case if the actual related description is missing. Moreover, knowledge representation of pre- and post-conditions in most works is done in an ad hoc manner.

2.7.4 Probabilistic approaches

These approaches are referred to as statistical analysis models, where alerts are correlated if they are statistically related. They are inspired from anomaly-based IDSs, where prior knowledge is not required. In this category, relationships between incurred events are computed statistically, providing automatic knowledge acquisition. In general, implementation of these approaches is performed using machine learning techniques. [29] proposed a combination of statistical and knowledge-base correlation techniques. Three algorithms are integrated based on the assumption that some attack stages have statistical and temporal relationships even though direct reasoning links are non-existent. A Bayesian-based correlation engine is used to identify the direct relations amongst alerts based on prior knowledge. In contrast to previous approaches, knowledge of attack steps is used as a constraint to the probabilistic inference. An engine based on Causal Discovery Theory is developed to discover the statistical of one-way dependence among alerts. In addition, a Granger Causality based algorithm is used by applying statistical and temporal correlation to identify mutual dependency. However, the problem of the selection of a time window for temporal correlation is still an unresolved problem. Attackers can exploit the slow-and-low attack to avoid detection. Attack prediction also relies on prior knowledge, and so 0-day attacks are not detected.

Recently, [37, 122-124] employed different data-mining algorithms for real-time correlation to discover multi-stage attacks. An offline attack graph is constructed using manual or automatic knowledge acquisition and then attack scenarios are recognized by correlating the collected alerts in real-time. The incoming step of an attack can be predicted after the detection of few attack steps in progress. In [122], an association rule mining algorithm is used to generate the attack graph from different attack classes based

on historical data. *Candidate attack sequences* are determined using a sliding window. In [124], an AprioriAll algorithm, which is a sequential pattern-matching technique, is used to generate correlation rules based on temporal and content constraints. [124] adopted a classical sequential mining method GSP (generalised sequential patterns) [125] to find the maximal alerts sequence and then to discover the attack strategy. The limitation of their work is the use of only attack class and temporal data as features.

Nevertheless, although these approaches do not require the construction of scenario rules by experts, a training dataset is needed. The dataset has to be collected and validated in order to obtain high-quality correlations. Therefore the required efforts to maintain a dataset are similar to the manual labour required to construct rules in other approaches. In addition, the false positives issue is another concern has to be taken in account; thus, these approaches can be utilized to support other techniques.

2.8 Alert verification

Generally, alert verification mechanisms are intended to distinguish successful from failed attacks. In typical deployments, the IDS device performs its detection producing a number of irrelevant alerts that have no effect on the target machine. That is because the host is not running the corresponding service or the service is not vulnerable. This knowledge gap between the IDS device and the protected system creates the issue of false positives [12]. The alert verification and vulnerability analysis problem has been investigated in several efforts at the IDS level [23, 126, 127]. Snort developers have brought up this point and have extended Snort to include facilities for adding configurable information about the target system. *Target-based analysis* has been introduced in Snort.2.8 [23] to model the targets rather than just the protocols. However,

this mechanism is limited to configurations based on information from OSs and these details have to be updated manually.

To deal with this issue, different techniques have been presented according to the context requirements. For instance, one direction is to compare a configuration file for the protected machines against the conditions required for the attack to be successful. The gathering of system configurations can be performed automatically and updated periodically using vulnerability scanners such as Nessus [128]. Other techniques are based on the analysis of the target system response after the attack occurs. This is typically performed as a further investigation required for forensics purposes.

[129] proposed M-Correlator to analyse and prioritize a stream of alerts and to verify relevant security incidents. The system is based on a knowledge base that contains a description for the protected network (topology and vulnerability) collected by the Nmap [130] tool. Three stages have been considered: 1) low-priority alerts are eliminated without preventing the IDS from generating them; 2) alerts are ranked using a *relevance score* based on a comparisons between topology and vulnerability information; and 3) alert priority is calculated according the significance of the target machine or service.

It should be noted that alert verification functionality should be employed as a lightweight process to avoid affecting overall system performance. Automatic and periodic knowledge acquisition is required to update stored data. This mechanism should be implemented as a complementary function to the IDS, and not integrated into the IDS itself. The reason behind this is to maximise the input data to the correlation system for deeper and more accurate analysis. Attack attempts should be recorded even if they are not successful because it may uncover some undetected activities.

2.9 Alert correlation system requirements

Although past techniques have dealt with reducing the massive number of collected data by NIDSs, there are many limitations. First, the analysis of attack strategy recognition is too complex, especially if the task is broadened to the prediction of unknown steps. Knowledge-based approaches are more accurate due to rule-matching mechanisms which are built based on expert knowledge, but they require more effort to provide precise rules. Statistical and temporal analysis techniques are unable to detect causal relations among events, but they do not require prior defined rules. The adoption of such systems in real-time is still an open question, where most proposed systems have been tested in an offline fashion or in a low-volume traffic environment. The huge number of detected events leads to graph explosion, as in [34, 35]. Moreover, missed attacks by the IDS can result in separate scenarios related to the same attack. Attackers also exploit the attack sliding window used in most approaches by performing slow-and-low attacks.

Alert correlation modelling has to provide a type of intelligence for attack strategy recognition. A framework consists of several components needed to make use of the capabilities of different approaches. Attack strategy recognition cannot be implemented in a single stage or by using a single component. In order to achieve this task, the correlation approach must consider:

- Real-time (or at least near real-time) correlation that inspects the incoming alerts and correlates them to the older ones. However, it is a challenging task, particularly if we consider the scalability, the huge amount of alerts and the speed of the current implementation of communication networks. [35] developed the TIAA system that performs the correlation in memory using a nested-loop mechanism, and [131] proposed

a queue graph mechanism. However, they have not provided any evaluation in high-speed networks to assess the system's scalability.

- Recognition of missed attack by the IDS, which will cause a division of a scenario or graph into separate ones. The correlation system has to be able to correlate isolated scenarios using implicit correlation. This mechanism can also be used to predict unknown attacks by hypothesizing about the expected step, which can consist of variations of known attacks.
- Slow-and-low attacks conducted by skilful attackers to avoid detection. Most of the implemented systems use a sliding window to avoid graph explosion, and hence very old events are ignored. However, determination of the value of the sliding window is also critical in order to provide a higher detection rate. Ignoring old events can result in the success of a dangerous intrusion attempt.
- Alert verification, where not all alerts are critical and where they have different effects on the system. This mechanism will reduce the huge number of correlated alerts by focusing on the significant ones.
- The configuration of the protected system can be incorporated in order to reduce false positives and to provide more meaningful and accurate results. Host response can also be involved to shift the focus to the critical events.

2.10 Conclusion

In this chapter, an overview of past and recent works in the field of IDSs and alert correlation techniques have been provided. A number of IDSs approaches have been discussed providing a historical summary to show the evolution achieved since this technology started. The two main IDSs methodologies have been investigated in details throughout this chapter presenting their advantages and disadvantages. We have also discussed the opportunity to exploit the capabilities of each approach by developing a

cooperative technique to employ both signature-based and anomaly-based IDSs. Moreover, host-based IDSs can be used to support the functionality of network-based IDS by providing further details about the protected system.

Alert correlation approaches have been employed to analyse alerts produced by IDSs to facilitate the detection of multi-stage attack. These mechanisms are used to reduce the huge amount of IDSs alarms and false positives. Different methodologies have been described in this respect including scenario-based, pre- and post-condition, and statistical methods. Then, we have stated the main requirements for alert correlation systems that have to be satisfied in order to develop a practical detection system.

CHAPTER 3: PERFORMANCE EVALUATION OF NETWORK INTRUSION DETECTION SYSTEMS (NIDS)

3.1 Introduction

Intrusion Detection Systems (IDSs) are designed for the security needs of networks. Existing Network Intrusion Detection Systems (NIDSs) are found to be limited in performance and utility, especially once subjected to heavy traffic conditions. An optimal methodology for the evaluation of NIDSs does not exist due to the heterogeneous nature of the operational environments. One aspect of NIDS evaluation is performance evaluation to measure the scalability of such systems in high-traffic environments. It has been observed that NIDS become less effective even when presented with a bandwidth of a few hundred megabits per second.

In this chapter, we have endeavoured to identify the causes leading to the unsatisfactory performance of NIDS. In this regard, an extensive performance evaluation of an open-source intrusion detection system (Snort) has been conducted. This has been done on a highly sophisticated test-bench with different traffic conditions. Host-based analysis and virtual-based analysis approaches have been selected to determine the performance of Snort. The performance of the system has been evaluated on different OS platforms (Windows, Linux and Free BSD) utilizing multi-core hardware. Our test methodology is also based on the concept of stressing the system and degrading its performance in terms of its packet-handling capacity. This has been achieved by: normal traffic generation, fuzzing, traffic saturation, parallel dissimilar attacks, manipulation of background traffic (e.g. fragmentation), packet sequence disturbance, and illegal packet insertion. Our results identified the performance limitations of Snort on both host and virtual platforms. We have also identified the factors responsible for the limited

performance of the system. Finally, we have discussed the factors involved in the limitation of IDSs performance.

3.2 NIDS evaluation

The design of a comprehensive approach to test and evaluate NIDS has been a debatable issue for many years. This is as a result of the nature of these systems running in heterogeneous environments and employing different detection methodologies. Host-based IDSs have testing requirements that are different from network-based IDS, and NIDSs themselves vary based on the employed operational techniques. However, several efforts in the literature review have been proposed to test and evaluate the performance and accuracy of these systems. Authors in [132] have presented a review of IDS evaluation methodologies by rendering available measurable characteristics of IDS testing. They have summarized criteria for IDS evaluation as follows:

1- *Detection coverage*: This measurement indicates the detection abilities of IDSs to recognise all known as well as potentially unknown attacks. IDS capabilities are measured by the maximum number of detected events – for instance, in signature-based methods, by performing a comparison between the number of signatures and known intrusive events. It is difficult to compare the different signature databases of various IDSs. In addition, a single database containing all known attacks does not exist. However, the Common Vulnerabilities and Exposures (CVE) [39] is a repository of publicly known vulnerabilities to enable information exchange between research and commercial products. However, the same vulnerability can be exploited by a set of attacks, and the value of detection varies from one environment to another.

2- *False positive and false negative rates*: While the detection capability is vital for IDSs, the false positive rate is also important, because overwhelming the system with a huge number of false alarms is impractical. False alarms can be produced by benign traffic such as network monitoring tools or by a signature that is not well defined. Various environments imply different network standards and different protocols and services, which cause difficulties in measuring the false positive rate. On the other hand, false negatives are caused by inability of NIDS to detect true attacks which are more serious than false positives. False negatives can be caused by improper written signatures, unpublicized vulnerability information, NIDS device is overloaded and cannot properly process all data, or poor NIDS device management. Reduction of false positives is not necessarily introducing false negatives if the implemented mechanism does not affect the detection coverage. Moreover, it is imperative to achieve a balance between false positives and false negatives. False positives affect productivity and false negatives affect security. Hence, it is essential to properly quantify risk and the NIDS role in risk reduction. False positives can be suppressed by different techniques: configuring the IDS to rely on the operational environment by tuning the signatures to only watch for specific services and operating systems, placing the NIDS behind the firewall, and alert analysis systems. The later one is the most secure and reliable technique that does not introduce in raise in false negatives rate. All signatures are enabled and the NIDS in configured with the maximum detection coverage. Alert correlation systems are the typical implementation of these techniques.

3- *Detection rate*: This measurement represents the detection accuracy; in other words, the IDS triggers a true alarm for the correct attack. In comparative evaluations, all

tested IDSs have to be configured in the same way and run in the same environment in order to obtain an accurate testing.

4- *Resistance to attacks*: Smart attackers can exploit weaknesses in the IDS itself to avoid detection with the use of several methods. They can stress the system by high-volume normal traffic to force the IDS to drop packets, or mutant crafted traffic packets can be injected to confuse signature-based systems and as a result, a huge number of false positives are generated.

5- *High-volume traffic handling*: The IDS's ability to handle higher traffic volumes is a critical issue in IDS evaluation. Sending a large amount of traffic – or even worse, high traffic with fragmentation, which is computationally expensive – can lead the IDS to collapse or at least drop packets. If the IDS starts to drop packets, this means that intrusive data can be passed to the protected system. Hardware-based NIDSs are more scalable than software-based systems for higher traffic.

6- *Event correlation abilities*: This is related to the context-based protection system, where the IDS is able to correlate information gathered from different sources. This information is valuable for building a state record for every event, and that allows the IDS to understand complex attacks such as multi-stage and hidden attacks.

7- *Detection of novel attacks*: This measurement is to determine IDS's ability to detect unknown or never-seen attacks. It suits anomaly-based approaches, which are capable of recognising novel attacks, but not signature-based approaches.

3.3 Background traffic

Despite having the criteria for the evaluation of IDS, the need for attack and realistic background traffic information remains in great demand. There are several approaches for obtaining attack and background traffic data. First, the most common and useful methodology is to build a test bed consisting of a number of connected machines and

other elements of a typical network infrastructure. This will simulate the real network despite the use of a limited number of hosts. The minimum number of hosts is three machines: 1) an attacker machine, containing a collection of exploit scripts or attacking tools, 2) a victim machine running vulnerable services, and 3) a monitoring machine running the IDS under testing. After the installation of the test bed, three kinds of traffic are required: normal traffic, background traffic and malicious traffic. The first type can be real traffic obtained from a production environment, or synthetic traffic generated by traffic generators [133-136], whether software-based or hardware-based. Malicious traffic is injected into the background network traffic, and this can be obtained from automated systems such as Metsploit [26] or the manual use of attack scripts. The advantage of this approach is the ease of traffic generation, the ability to repeat the test many times, and the fact that the traffic can be recorded and distributed publicly. On the other hand, this method is expensive and time consuming, particularly if commercial traffic generators [135, 136] are considered, as they perform better than the few, not well documented open-source software ones. Moreover, the use of a limited number of hosts implies less running services, less implemented protocols and the absence of huge number of concurrent connections. Synthetic traffic is generated based on random variables, and some IDSs consider this type of traffic abnormal and may ignore it.

Second, real traffic obtained from a production network infrastructure can be used for IDS evaluation to offer a more realistic approach. The IDS system is connected to a tap or a mirrored port on the edge of a network. This method is less common for reasons of privacy and due to the difficulty in identifying potential unlabeled attacks. Some traffic generators such as Harpoon [137] have been developed as intelligent generators by obtaining real traffic characteristics from live networks.

Third, real traffic can be modified to remove all sensitive data and used for testing of IDS systems, which is called *sanitised traffic*. However, the main task for the IDS is to inspect content and attacks usually residing in packet payload, so this is not the optimal approach for testing IDSs, but can be suitable for other network components.

The first well-documented IDS evaluation methodology to be introduced was the DARPA evaluation 1998 (UNIX dataset) developed in the labs of MIT [55], followed by another dataset in 1999 (Windows NT dataset). Data used in their experiments are labelled and distributed publicly. DARPA datasets have been criticised for not being updated since 1999, for some of the attack types used having become obsolete, and for not covering new emerging attacks [22]. The Lincoln Adaptable Real-time Information Assurance Test-bed (LARIAT) [138] is another evaluation methodology providing a tool for simulation and testing. DEFCON [54] is the worldwide hacker and security expert conference and competition. Malicious traffic can be obtained from DEFCON CTF (Capture The Flag), which contains a huge number of attacking traffic used for IDS testing. NSS Labs [41] is a commercial group that provides a comprehensive methodology for the evaluation of NIDSs. Their approach includes security effectiveness, performance, resistance to evasion techniques, stateful operation, latency, reliability and usability [41]. Background traffic is generated from hardware-based traffic generators such as Spirent SmartBit [136]. Malicious traffic is obtained from automatic tools such as Metasploit [26] and CANVAS [139], or manually defined attacks.

Other efforts have been made to test signature-based IDSs by analysing the collection of attack signatures and then generating mutant patterns to hide the actual attacks. The authors in [25] have developed a cross-testing approach to generate synthetic events to

test IDS ability and identify real attacks from modified ones similar to the signatures. [140] addressed the need for publicly well-documented datasets for IDS testing. He presents a set of tools that generate malicious traffic using a virtual network infrastructure. Different platforms were used to create attack traces against various OSs and violating different system services. [141] proposed a framework for offline and online testing to evaluate NIDS resistance to evasion techniques. A comparative evaluation methodology was presented to test Snort and Bro by generating ambiguities in traffic traces.

3.4 Motivation

A typical scenario of employing a NIDS in a network is its implementation on the server with minimum active services. This setup is quite susceptible to insider attacks, especially in high-speed environments. The current NIDSs are also threatened by resource crunch attempts such as DDoS, which has increased from a few megabits in the year 2000 to 40 Gbps in 2008 [142]. The performance criteria of NIDSs demand that every single packet (header, payload) passing through the network needs to be evaluated with the same link speed; however, the massive increase in network speed has generated many concerns. Sending a large amount of traffic or using computationally expensive techniques like fragmentation can compromise a NIDS or make it to start dropping packets.

NIDSs can be implemented as software-based or hardware-based. Software-based NIDSs are more configurable, easy to update and need less maintenance; however, their performance is quite slow. On the other hand, hardware-based NIDSs can handle a larger volume of traffic, but they are expensive, require more maintenance and are hard to update. The choice between the two is a trade-off between cost and performance.

This has created the need to evaluate the current software-based systems. This is especially so in current-day high-speed conditions using different implementation scenarios.

We have identified that quite few efforts have been made to measure the performance of NIDSs. Most of the evaluation methodologies are based on testing in moderate traffic conditions. Furthermore, some of these approaches have used previously saved datasets rather than real traffic. These seem unrealistic, as actual system performance was gauged under limited conditions with non-realistic network flow. The results obtained under these conditions could not portray the actual performance output. We have endeavoured to evaluate the system against realistic network conditions, providing the application with different tiers of hardware support in order to analyze its performance more practically. The recent development of multi-core systems has also added a few more opportunities for deploying a software-based system; these shall also be investigated in this chapter.

Our aim in this chapter is to provide answers to the following questions:

- Is it possible to deploy a current software-based NIDS such as Snort at a rate above 500 Mbps using commodity hardware? Also to identify the limits of incoming traffic, a system can handle effectively in terms of packet loss.
- Does the use of different OSs (normal desktop, server), hardware capabilities (single, multi-core) and configurations (host, virtual) affect NIDS performance?
- Identification of mechanisms to improve NIDS performance in high-speed traffic before shifting to hardware solutions.

It is essential that the NIDS is capable to process packets traverse the protected network with speed of the communication link [6, 13, 14]. When the network traffic load

becomes higher than the peak processing throughput the NIDS can sustain, the CPU becomes saturated, and the Operating System inevitably starts dropping packets before delivering them to the NIDS, impeding its detection ability [15-17]. Since these packets are not inspected, if they are part of an attack or other malicious activity, then that event will be missed[27, 45].

Assuming a uniform distribution of packets across the network traffic, any packet loss results in a proportional loss in NIDS effectiveness [11]. This relationship has been widely identified in NIDS research [143-146]. Figure 3.1 illustrates the relationship between packet loss and missed alert rate which consequently cause missing attacks and affect the NIDS precision. The scatter plot shows a direct and nearly a linear relationship between the two parameters. The number of missed alerts approaches zero if the packet loss percentage becomes small. The network traffic used in this experiment consists of 530,000 packets containing 521 attacks (1000 packets/alert).

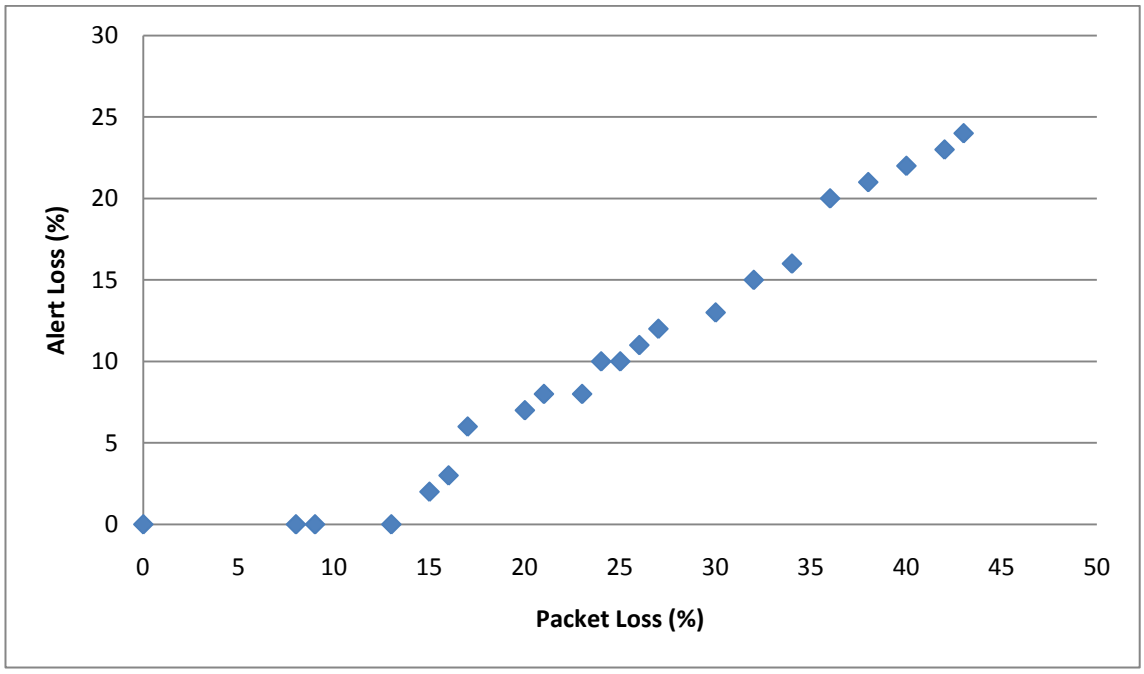

Figure 3.1 Relationship between packet loss & missing alerts

Our research has focused on signature-based IDSs with an emphasis on evaluating their performance in high-speed traffic conditions. Snort has been selected as a test platform because of its popularity and status as a de facto IDS standard. We are confident that the results obtained in this research would be equally applicable to other IDSs available on the market. The test environments selected for the research have a significant edge over [40], and our results develop a new understanding of IDS performance limitations.

3.5 Evaluation Methodology

Our evaluation methodology is based on the concept of analyzing the system capacity in terms of its packet-handling capability by implementing it into different hardware configurations and testing platforms. This has been achieved by establishing three different test-benches, where every test-bench has been assigned a specific evaluation task. Test-bench 1 implements the Snort on mid-range commodity hardware (limited processing power and system memory). The results obtained on this platform describe the efficacy of NIDS implementation at this level. Test-benches 2 and 3 utilize high-range commodity hardware built on an Intel Xeon Dual Quad-Core processor using 4.0 GB RAM. These test-benches analyzed the system performance on host and virtual configurations respectively. The system capability has been also analyzed by observing its response to known attacks in Test-bench 1; however, this criterion has not been considered for other test-benches due to lack of space. Table 3.1 summarizes the three test benches and the parameters set for each test bench.

In the initial phase of the research, the aim was to measure the performance of Snort installed on normal machine and using host and virtual configuration. Both normal and attack traffic are injected in the testing network to evaluate the detection coverage. The obtained results from test bench 1 have showed a proportional relationship between

capability of packet processing and detection coverage. The inability of Snort to handle all received packets online is a direct cause to the low rate in detection capacity. Consequently, the focus has been shifted from evaluating the detection coverage to the capacity of packet handling. For this reason, in test bench 2 and test bench 3 , attack traffic is not considered because missing a packet carrying attack evidence leads to missing the corresponding alert. In addition, to provide a precise measurement of detection coverage of any IDS, it is necessary to insure that all other related factors have no effect.

Table 3.1 Summary of test benches.

Test bench	**Host config.**	**Virtual Config.**	**Resourceful machines**	**Traffic**	
				Normal	**Attack**
Test bench 1	Yes	Yes	No	Yes	Yes
Test bench 2	Yes	No	Yes	Yes	No
Test bench 3	No	Yes	Yes	Yes	No

Snort IDS has been selected for our testing for being an open source and the de facto standard for IDS/IPS. It is the most widely deployed intrusion detection and prevention technology worldwide. It has the most numerous and active community in the open source NIDS field today. In addition, several commercial products use Snort as their core technology and Snort signatures are included in many industry security systems. As a network device, Snort has been developed to be a lightweight system and fast in order to keep up with increasing network bandwidths. Moreover, Snort is flexible and can be used in different ways from a simple network sniffer to true gateway IDS. It is configurable, its signatures can be customized and developed easily, and its inner working can be modified according to the operation environment. In contrast, other

open sources IDS platforms such as Bro, lack of an up-to-date set of signatures and lack of full support and product documentation.

3.6 Test-bench 1

The network is composed of six machines using a Pro-Curve Series 2900 switch [147], as shown in Figure 3.2. The test-bench comprises a number of high-performance PCs running open-source tools to generate background traffic, run attack signatures and monitor network performance.

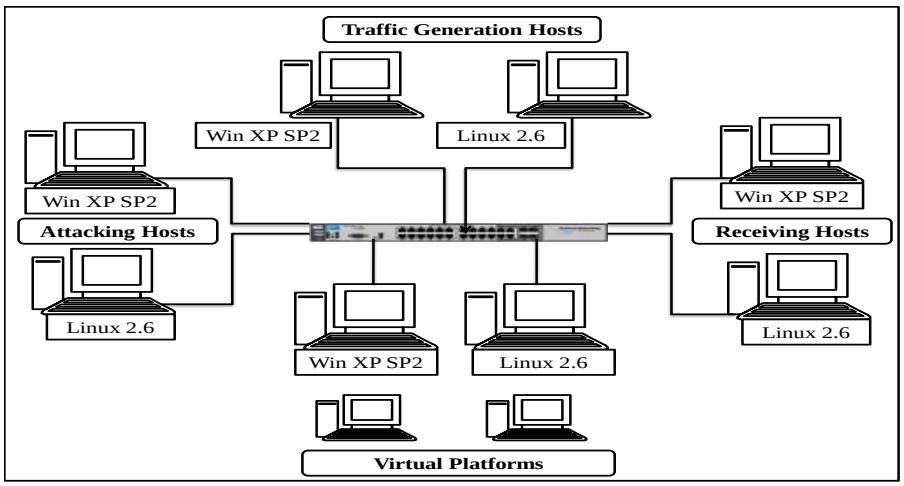

Figure 3.2 Test Bench-1.

3.6.1 Hardware description

The hardware description of the network is shown in Table 3.2. The network components are described as follows:

Traffic generators

Two machines are configured to generate network traffic on Windows XP SP 2 and Linux 2.6, respectively, as shown in Figure 3.2. The distribution of network traffic is TCP (70%), UDP (20%) and ICMP (10%).

Attacking host

Two machines are configured to generate attacks/exploits on Windows XP SP 2 and Linux 2.6, as shown in Figure 3.2.

IDS machine (Snort)

In the test-bench, Snort is operated on both host and virtual machines for both Windows and Linux platforms. This has been done to analyze the performance of Snort using the limited resources of a virtual machine as well as with the full processing capability of a host computer. Snort version 2.8.3 [23] has been selected for evaluation.

Table 3.2 Network Description – Test-bench 1.

Machine Type	**Hardware Description**	**Tools Used**
Network traffic/ back ground traffic generator (Win XP SP2)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbps RJ45, Network Card (Broadcom NetXtremo Gigabit Ethernet).	Traffic Generators: NetCPS [148], Tfgen[149], Http Traffic Gen [150], LAN Traffic Version 2 [134] and D-ITG Version 2.6 [133]
Network traffic/ back ground traffic generator (Linux 2.6)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbpss RJ45, Network Card (Broadcom NetXtremo Gigabit Ethernet).	Traffic Generators: D-ITG Version 2.6 [133] and hping Version 2 [151]
Attack Machine • Win XP SP2 • Linux 2.6	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbps RJ45, Network Card (Broadcom NetXtremo Gigabit Ethernet).	Attacking tool: Metasploit framework [26]
IDS Machines • Snort – Win XP SP2 • Snort – Linux 2.6	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbps RJ45, Network Card (Broadcom NetXtremo Gigabit Ethernet).	• IDS:Snort [23], Traffic Monitor: Bandwidth Monitor [152] on Win XP SP2 • IDS:Snort and Traffic Monitor: nload [153] on Linux 2.6.
Switch	ProCurve Series 2900 , 10 Gbps switch with 24x1 Gbps ports and 2x10 Gbps 3CR17762-91-UK ports.	

Snort was also tested for its accuracy on the different OS platforms (Windows and Linux). The platforms were tested by injecting a mixture of heavy network traffic and scripted attacks through the Snort host. *Snort.conf* file in its default configuration was selected for evaluation. The performance of Snort was also evaluated under the following variant conditions:

- Generating attacks from different OS hosts.
- Varying traffic payload, protocol and attack traffic in different scenarios, as shown in Table 3.3.
- Subjecting it to hardware constraints of virtual machine configurations.

Table 3.3 Test-bench 1 scenarios.

Scenario	Network Traffic (PC 1)	Network Traffic (PC 2)	Attack Machine (Metasploit)	IDS Machine (Snort)
Alpha	Host Windows	Host Windows	Host Linux 2.6	Virtual Windows
Bravo	Host Windows	Host Windows	Host Linux 2.6	Virtual Linux 2.6
Charlie	Host Windows	Host Windows	Host Linux 2.6	Host Windows
Delta	Host Windows	Host Windows	Host Linux 2.6	Host Linux 2.6
Echo	Host Windows	Host Windows	Host Win	Host Linux 2.6

3.6.2 Results

Snort was evaluated on the basis of network traffic ranging from 100 Mbps to 1.0 Gbps (divided into five different test scenarios). The other parameters selected for evaluation include network utilization, CPU usage and Snort CPU usage. Snort performance in terms of packets analyzed, packets dropped, alerts/logs and detection statuses have also been considered for critical evaluation.

3.6.2.1 Scenario Alpha

Snort was configured to run using the performance-limiting configuration of a Windows XP SP 2 virtual machine. It was subjected to heavy background traffic and attack exploits (from a well-resourced Linux host). The results obtained are shown in Figure 3.3. They demonstrate that the performance of Snort deteriorates markedly as network traffic load increases.

3.6.2.2 Scenario Bravo

Snort was configured to run using the performance-limiting configuration of a Linux virtual machine and the attacker was a well-resourced Linux host. The results obtained, as shown in Figure 3.4, identify similar performance limitations as found in Scenario Alpha. However, an improvement can be observed when Snort runs on the same OS as that of the attacking host.

Attack Platform: Host Linux 2.6 **vs** Snort Platform: Virtual Windows

Parameter	100 – 200 Mbps	500 – 700 Mbps	800 Mbps – 1.0 Gbps
Network Utilization	12 %	56%	90%
CPU Usage	50 – 70%	90 – 100%	95 – 100%
Snort CPU Usage	40 – 50%	80 – 90%	90%
Packets Analysed	72.5%	66%	38 %
Packets Dropped	27.5%	34%	62 %
Alerts & Logged	83%	62%	28%

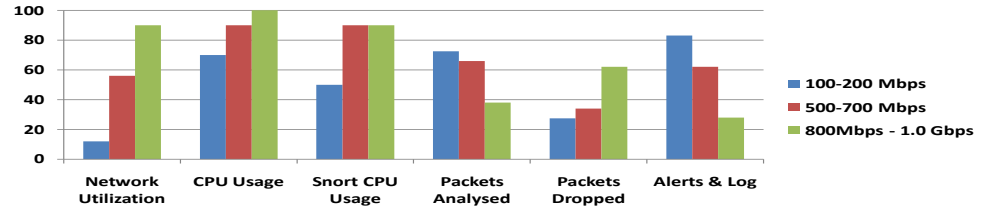

Figure 3.3 Results – Scenario Alpha.

Attack Platform: Host Linux 2.6 **vs** Snort Platform: Virtual Linux 2.6

Parameter	100 – 200 Mbps	500 – 700 Mbps	800 Mbps – 1.0 Gbps
Network Utilization	12 %	54 %	90%
CPU Usage	50 – 70%	88 - 95%	90 – 100%
Snort CPU Usage	40 – 50%	75 - 85%	90-95%
Packets Analysed	75 %	62 %	45%
Packets Dropped	25 %	38 %	55 %
Alerts & Logged	85%	64 %	36 %

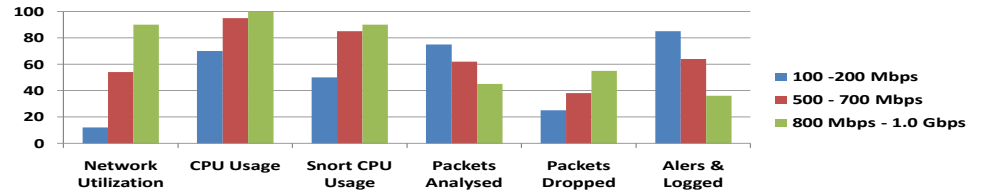

Figure 3.4 Results – Scenario Bravo.

3.6.2.3 Scenario Charlie

Snort was configured to run using a well-resourced Windows platform with the attacker on a Linux host. The results obtained are shown in Figure 3.5. Snort performance declines as a result of being run on a different OS platform to that of the attacker. However, an improvement can be observed in comparison to the equivalent virtual scenario.

Attack Platform: Host Linux 2.6 **vs** Snort Platform: Host Windows

Parameter	100 – 200 Mbps	500 – 700 Mbps	800 Mbps – 1.0 Gbps
Network Utilization	13%	53%	90%
CPU Usage	20 – 30%	30 - 35%	35 – 40%
Snort CPU Usage	15 – 20%	20 - 25%	25-30%
Packets Analysed	98.2 %	38 %	27 %
Packets Dropped	1.8 %	62 %	73 %
Alerts & Logged	100%	47 %	24 %

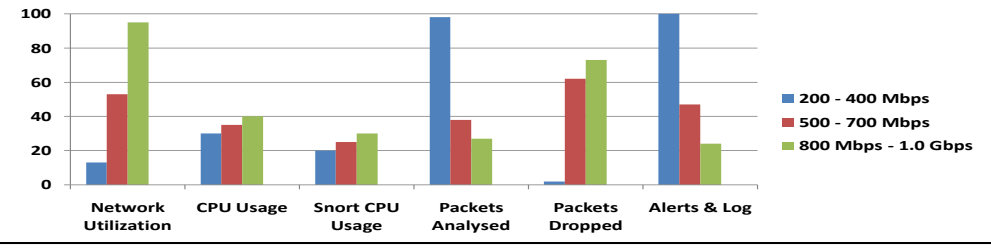

Figure 3.5 Results – Scenario Charlie.

3.6.2.4 Scenario Delta

Snort and the attacker were both configured using a well-resourced Linux platform as host. The results obtained are shown in Figure 3.6. Comparatively, an improved performance for Snort can be observed in this scenario, as both attacker and Snort are using the same OS (Linux).

Attack Platform: Host Linux 2.6 **vs** Snort Platform: Host Linux 2.6

Parameter	100 – 200 Mbps	500 – 700 Mbps	800 Mbps – 1.0 Gbps
Network Utilization	21%	55%	95%
CPU Usage	18 – 25%	29 - 36%	38 – 43%
Snort CPU Usage	15 – 20%	22 - 27%	29-36%
Packets Analysed	98.5%	47 %	39 %
Packets Dropped	1.5%	53 %	61 %
Alerts & Logged	100%	67 %	33 %

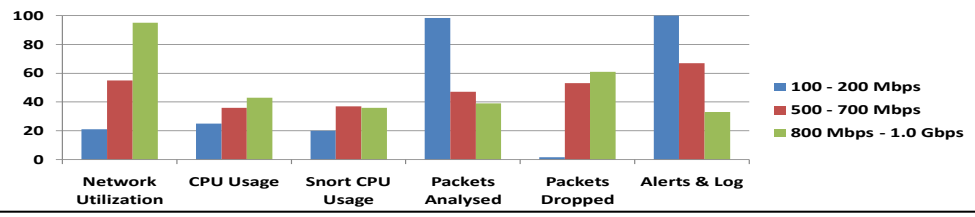

Figure 3.3 Results – Scenario Delta.

3.6.2.5 Scenario Echo

Snort is configured to run on a well-resourced Linux platform and the attacker on a Windows host. The results obtained are shown in Figure 3.7. Similar results were

obtained to those in Scenario Charlie, where the OS platform used Snort and attacker are reversed.

Attack Platform: Host Windows **vs** Snort Platform: Host Linux 2.6

Parameter	100 – 200 Mbps	500 – 700 Mbps	800 Mbps – 1.0 Gbps
Network Utilization	15%	54 %	96%
CPU Usage	25 – 30%	32 - 35%	38 – 45 %
Snort CPU Usage	18 – 22%	22 - 26%	27-35%
Packets Analysed	99 %	42 %	35 %
Packets Dropped	1 %	58 %	65 %
Alerts & Logged	100%	65 %	35 %

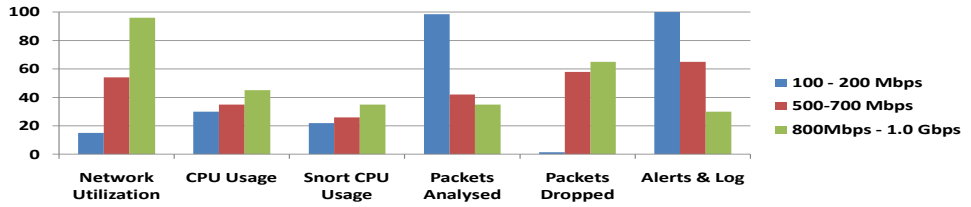

Figure 3.7 Results – Scenario Echo.

3.7 Test-bench 2

Snort has been implemented on a fully resourceful host machine built on a dual quad-core processor using 4.0 Gb RAM. The configuration of the network machines are shown in Table 3.4.

Table.3.4 Network description – Test-bench 2 and Test-bench 3.

Machine Type	Hardware Description	Tools Used
Network traffic/ back ground traffic generator (Win XP SP2)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbps RJ45, Network Card (Broadcom NetXtremo Gigabit Ethernet), L2 Cache 2 x 4.0 MB, FSB 1066 MHz.	Traffic Generators: NetCPS [148], Tfgen [149], Http Traffic Gen [150], LAN Traffic Version 2 [134] and D-ITG Version 2.6 [133]
Network traffic/ back ground traffic generator (Linux 2.6)	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbps RJ45, Network Card (Broadcom NetXtremo Gigabit Ethernet), L2 Cache 2 x 4.0 MB, FSB 1066 MHz.	Traffic Generators: D-ITG Version 2.6 [133] and hping Version 2 [151]
IDS Machine	Dell Precision T5400, Intel Xeon **Dual Quad-Core 2.0 GHz,** 4 GB RAM, L2 cache 2x6 MB, FSB 1066 MHz, PCIe, Network Interface Card, 10 Gbps Chelsio, HD: 1000 GB, Buffer 32 MB, SATA.	IDS: Snort[23]
Receiving Hosts • Win XP SP2 • Linux 2.6	Dell Precision T3400, Intel Quad-Core, Q6600 2.40 GHz. 2 GB RAM, PCIe, 1.0 Gbps RJ45, NIC 10 Gbps Chelsio on Win XP SP2 host and Linux 2.6 host has Broadcom NetXtremo Gigabit Ethernet.	• Win XP SP2 – LAN Traffic Generator • Linux 2.6 – D-ITG Traffic Generator
Switch	ProCurve Series 2900 , 10 Gbps Switch with 24x1 Gbps ports and 2x10 Gbps 3CR17762-91-UK ports.	

Figure 3.8 describes the test-bench where Snort been respectively evaluated on the fully resourceful platforms built on Windows Server 2008, Linux Server 2.6 and Free BSD 7.0. The system's performance is gauged in terms of its packet-handling capacity of the application built on respective platforms for different types of network traffic.

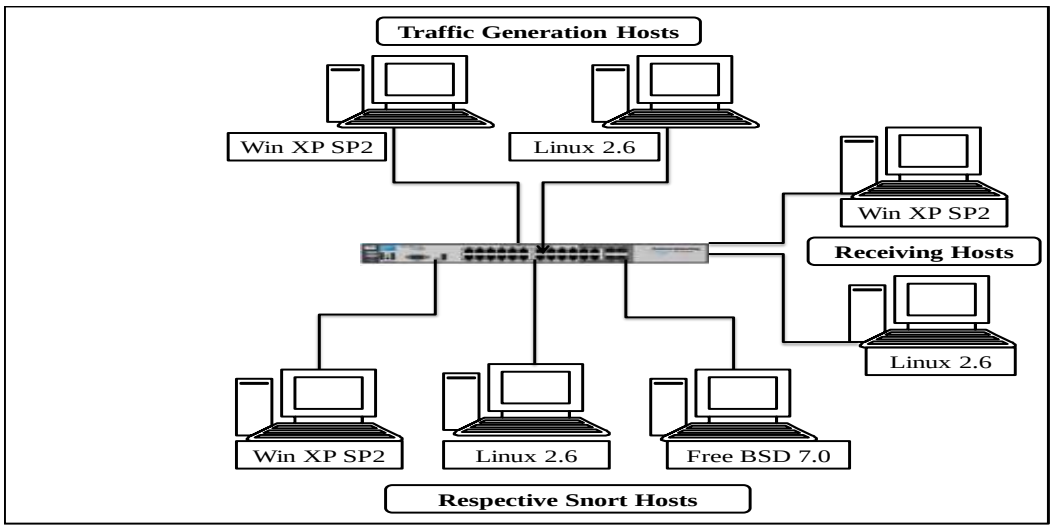

Figure 3.8 Test-bench 2 – Host configuration

3.7.1 Evaluation methodology

- Different packet sizes (128, 256, 512, 1024 and 1514 bytes) were generated, and Snort's performance at the following traffic loads was evaluated: 750Mbps, 1.0 Gbps, 1.5 Gbps and 2.0 Gbps, respectively.
- Varying traffic payload: UDP and mixed TCP, UDP and ICMP traffic.
- Snort's performance characteristics were evaluated – packets received, packets analysed, packets dropped and CPU usage – at various packet sizes and bandwidth levels.
- Duration of test: 1, 5 and 10 minutes, where the average value of the results obtained has been taken.

3.7.2 Results

The response of the IDS (Snort), i.e. dropped packets, on UDP traffic injected in various packet sizes and bandwidths is shown in Table 3.5; each scenario is discussed in the following paragraphs:

Table 3.5 Host-based configuration results (Packets dropped(%)) – UDP traffic.

traffic	OS	128B	256B	512B	1024B	1514B
750 MB	FreeBsd	15.4	9.45	3.29	6.64	6.26
	Linux	56.91	52.67	27.83	6.72	6.4
	Windows	51.76	50.62	25.32	6.83	6.35
1 G	FreeBsd	52.6	32.15	28.4	25.04	24.89
	Linux	72.7	69.04	65.88	55.26	53.35
	Windows	68.05	66.82	61.97	53.6	52.9
1.5 G	FreeBsd	66.7	62.03	46.22	41.6	40.8
	Linux	77.6	71.5	67.32	57.1	55.5
	Windows	80.6	74.7	70.23	68.31	64.6
2 G	FreeBsd	74.07	69.8	65.3	50.54	49.4
	Linux	78.04	75.8	69.6	59.3	57.3
	Windows	93.5	91.0	88.85	77.5	70.8

3.7.2.1 UDP traffic

i. UDP traffic – 750 Mbps. The Performance of all OSs linearly improved from smaller packet sizes (128 Bytes) to larger ones (1514 Bytes); however, Free BSD shows a significant edge over the others in all ranges of packet sizes, as shown in Figure 3.9.

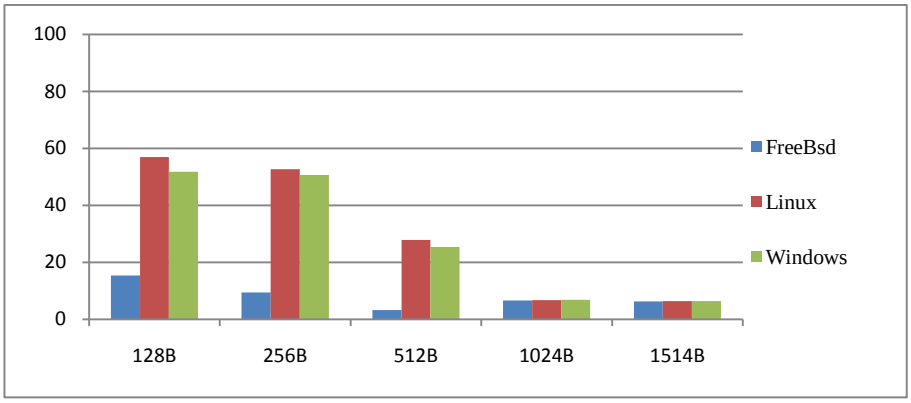

Figure 3.9 Results: packet dropped, UDP traffic – 750 Mbps.

ii. UDP traffic– 1.0 Gbps. Increase in the bandwidth shows a decline in the performance of the system, resulting in more packet loss. A considerably uniform response has been observed in all categories of packet sizes from all platforms tested.

This scenario also showed a comparatively improved (though not ideal) performance for Free BSD as shown in Figure 3.10 .

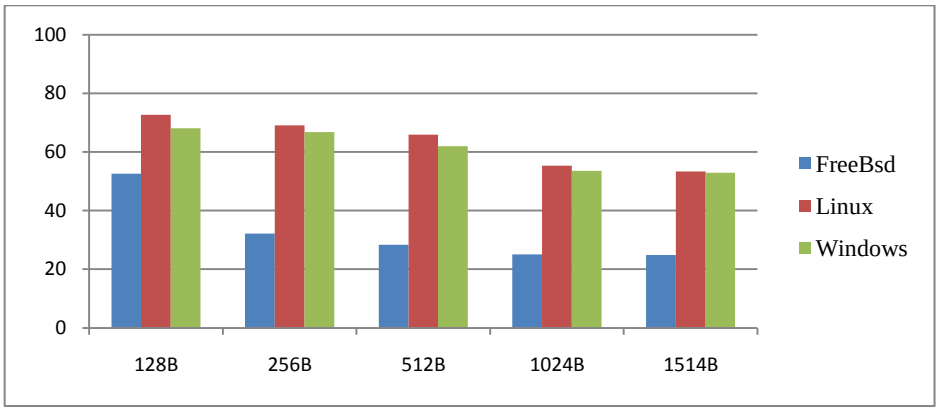

Figure 3.10 Results: packets dropped, UDP traffic – 1.0 Gbps.

iii. UDP traffic – 1.5 Gbps. A further increase in the traffic bandwidth resulted in higher packet loss by the system. Approximately similar performances were observed for all packet sizes, the response indicating that Free BSD performed better, followed by Linux, and then by Windows in last place as shown in Figure 3.11.

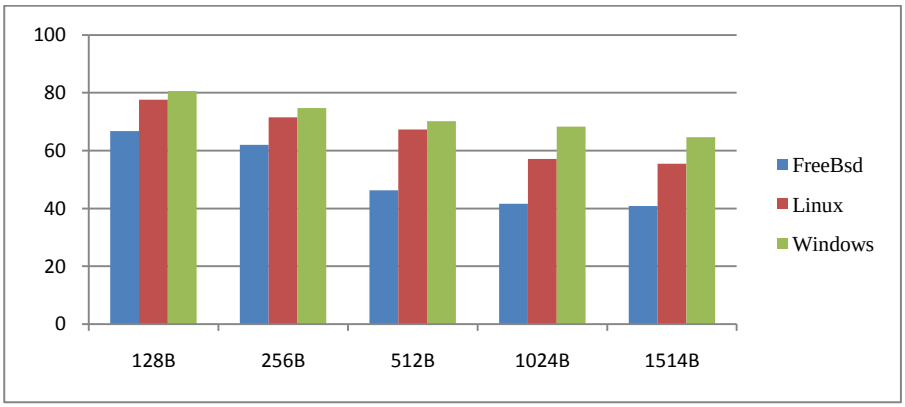

Figure 3.11 Results: packets dropped, UDP traffic – 1.5 Gbps.

iv. UDP traffic – 2.0 Gbps. At 2.0 Gbps of traffic input, the performance of Windows seemed totally compromised at 128 Bytes of packet sizes. The platform lost virtually all the input traffic and performed no evaluation. The performance gradually increases for higher packet sizes, in a similar pattern as that observed for the lower traffic bandwidths as shown in Figure 3.12. This, however, displayed a highly compromised performance for all platforms, identifying strong limitations in handling input traffic reaching 2.0

Gbps. In practice, system built on Free BSD, Linux and Windows platforms once subjected to 2.0 Gbps of input traffic suffer heavy packet loss.

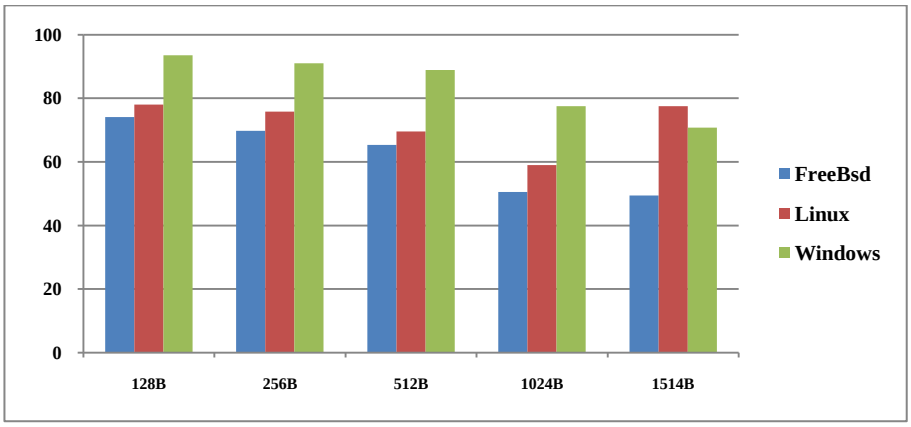

Figure 3.12 Results: packets dropped, UDP Traffic – 2.0 Gbps

3.7.2.2 Mixed traffic

The mixture of TCP (70%), UDP (20%) and ICMP (10%) traffic was generated replicating realistic network flow as follows:

- Generating random packet sizes and observing system response – packet handling capacity.
- Traffic bandwidth limited to 1.0 Gbps – supporting commodity hardware on account of system implementation as a test-bench.
- Recording packet drop statistics for all three Snort platforms built on Free BSD, Linux and Windows respectively.

The main reason to conduct this test is to ascertain the performance of a system under realistic network conditions. The results here also followed quite similar patterns of system response. Table 3.6 describes the results obtained. Free BSD showed quite good performance in terms of handling mixed traffic for the bandwidth of 1.0 Gbps on a multi-core implementation.

Table 3.6 Host-based configuration results – mixed traffic

Operating System	Dropped Packets%
FreeBSD	21.7
Linux	27.2
Windows	26.3

3.8 Test-bench 3

Virtualization is a framework for abstracting the resources of a computer into multiple execution platforms by creating multiple machines on a single computer. Each machine operates on the allocated hardware and can afford multiple instances of applications [154]. This concept has been successfully incepted within the industry/business community. The mechanics of system virtualization for the implementation of network security tools have been considered appropriate by academics in the field of information security [155].

The concept has been developed to address issues relating to the reliability, security, costs and complexity of the network/systems. It has successfully been used for the processing of legacy applications, ensuring load balancing requirements, resource sharing and tasking among virtual machines by using autonomic computing techniques. The technique has also shown merits in the situation where an application failure on one machine does not affect the other. In addition, ease of isolation allows multiple OS platforms to be built on one machine running variable instances of applications. This has made the concept quite fascinating for the research community [156]. The test-bench is distributed into three parts and configured around a ProCurve series 2900 switch, as shown in Figure 3.13.

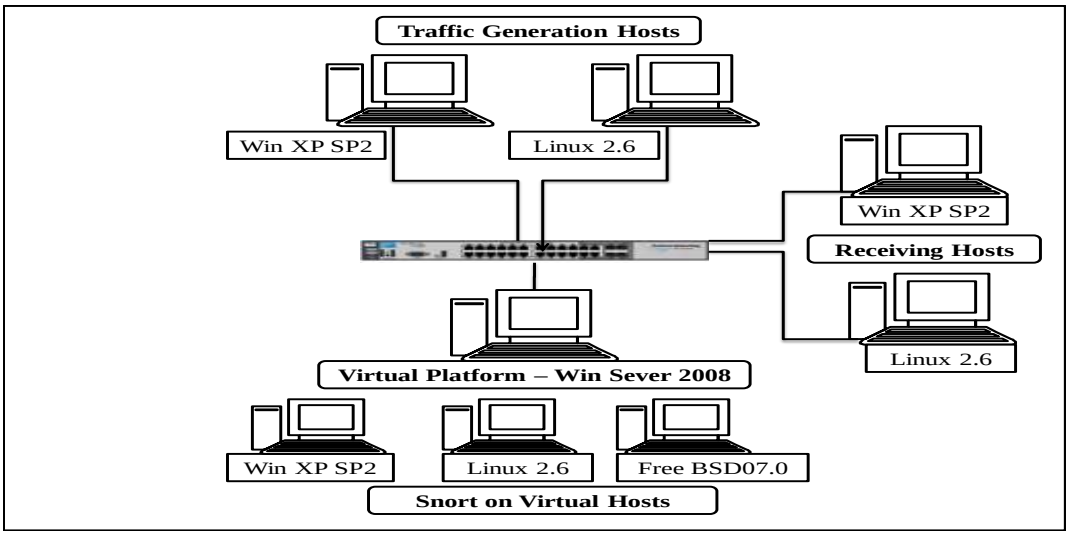

Figure 3.13 Test-bench 3 – Virtual configuration.

The basic idea of the evaluation process revolves around packet capturing and evaluation by virtual platforms and Snort. Two machines for traffic generation have been selected: Linux 2.6 and Windows XP SP2 platforms. Similarly, the traffic reception machines were also deployed to fulfil network requirements. Details of the traffic generation tools are shown in Table 3.4.

The virtual platform running Snort has been configured on a dual quad-core processor. The machine hardware details are listed in Table 3.4. The system is built on the Windows 2008 Server platform and three separate virtual platforms have been created Windows XP SP2, Linux 2.6 and Free BSD 7.1. Snort is running simultaneously on all the virtual machines and similar traffic loads and types are injected onto all platforms.

3.8.1 Evaluation methodology

In order to ascertain the capability of Snort to handle high-speed network traffic on virtual platforms, we proceeded as follows:

- Parallel Snort sessions were run on all virtual machines.
- The machines were injected with similar traffic-load characteristics (UDP and TCP traffic) for 10 minutes.

- Different packet sizes (128, 256, 512, 1024 and 1460 bytes) were generated and Snort's performance at the following traffic loads was evaluated: 100 Mbps, 250 Mbps, 500 Mbps, 750 Mbps, 1.0 Gbps and 2.0 Gbps, respectively.
- Snort's performance characteristics were evaluated – packets received, packets analysed, packets dropped, and CPU usage at various packet sizes and bandwidth levels.
- Packets received were compared at both the host OS and the virtual platforms running the Snort applications.
- During the course of the tests, no changes were made in OS implementation, specifically Linux using NAPI- MMMP[1] and Free BSD using PF-RING -BPF[2] [156].

3.8.2 Results

The results are distributed over UDP and TCP traffic types respectively. It was observed that the total packets transmitted from the traffic-generating PCs was equivalent to the number of packets received at the host machine/OS running virtual platforms, as shown in Table 3.7; however this was not the case once the system was found[3] non-responsive.

Table 3.7 Packets received at host OS.

Total Packets Received at OS (Millions) – UDP					
Bandwidth	**128 Bytes**	**256 Bytes**	**512 Bytes**	**1024 Bytes**	**1460 Bytes**
100 MB	60	35.82	17.77	10.56	6.96
250 MB	178.1	94.14	48.00	18.34	20.22
500 MB	358.3	148.29	92.56	46.2	39.00
750 MB	System Non Responsive		144.72	91.56	45.23
1.0 GB	System Non Responsive			167.40	78.00
2.0 GB	System Non Responsive				

Total Packets Received at OS (Millions) – TCP			
Bandwidth	**50 Connections**	**100 Connections**	**200 Connections**
100 MB	10	26.7	21.60
250 MB	31.86	39.763	48.69
500 MB	67.90	108.56	84.098
750 MB	80.29	113.72	124.58
1.0 GB	102.51	118.144	148.982
2.0 GB	147.54	170.994	221.28

[1] Modified device driver packet handling procedures.
[2] Berkley Packet Filter.
[3] In non-responsive situations we consider 100% packet loss.

3.8.2.1 UDP traffic

The results below are described in relation to packet size, bandwidth (i.e. traffic load), and the virtual OS platform running the Snort application:

i. Snort response for packet sizes of 128 and 256 Bytes

- Linux shows quite good performance for these packet sizes up to 250 Mbps of traffic load; its performance declined at higher bandwidth levels, as shown in Figure 3.14. The system was found non-responsive at traffic loads of 750 Mbps and above.
- Windows shows good performance for 128 Bytes packet sizes at 100 Mbps loading only. Its performance is compromised at higher loading levels, as shown in Figure 3.14. The system was also found non-responsive at traffic loads of 750 Mbps and above.
- Free BSD performs slightly better than Windows, as shown in Figure 3.14. The system was also found non-responsive at traffic loads of 750 Mbps and above.

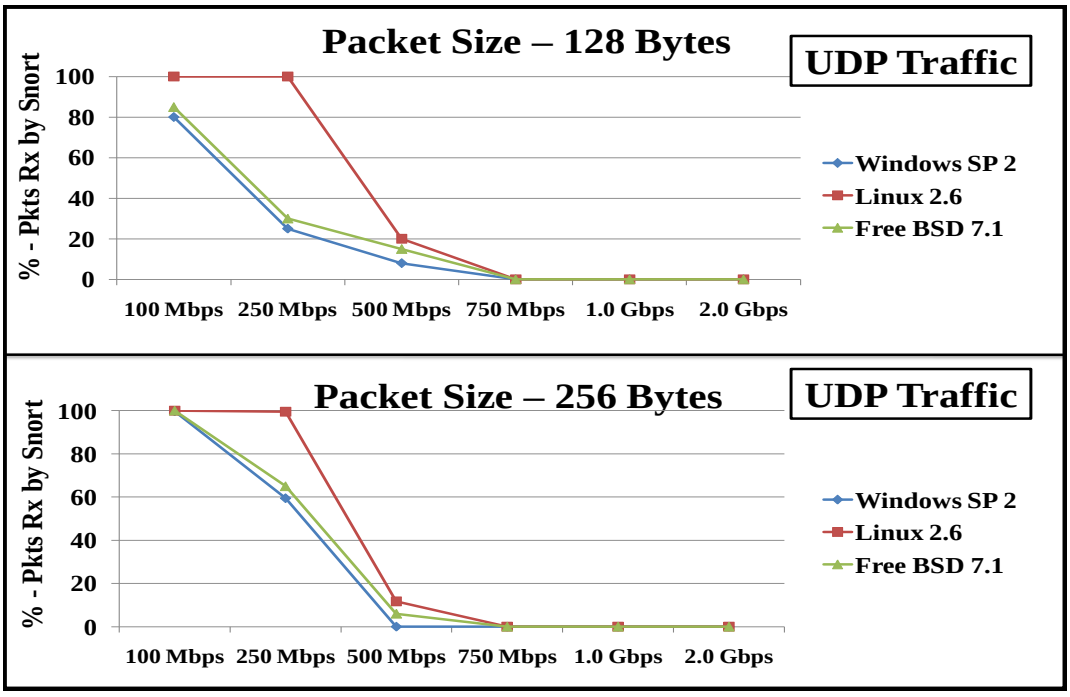

Figure 3.14 Snort packets received (%) – UDP traffic (128 Bytes & 256 Bytes).

ii. Snort response for packet sizes of 512 and 1024 Bytes

- Linux shows quite good performance for traffic loads of up to 500 Mbps for all packet sizes, as shown in Figure 3.15. However, the Linux system was found non-responsive at traffic loads of 1.0 Gbps and above for 512 Byte packet sizes, and at 2.0 Gbps for packet sizes of 1024 Bytes.

- Windows also performed satisfactorily at traffic loads of 250 Mbps and 500 Mbps for packet sizes of 512 Bytes and 1024 Bytes respectively, as shown in Figure 3.15. The system found non-responsive at traffic loads of 1.0 Gbps and above for packet sizes of 512 Bytes, and 2.0 Gbps for packet sizes of 1024 Bytes.

- Free BSD responds a bit better than Windows, as shown in Figure 3.15. The system was found non-responsive at traffic loads greater than 1.0 Gbps for packet sizes of 512 Bytes, and 2.0 Gbps for packet sizes of 1024 Bytes.

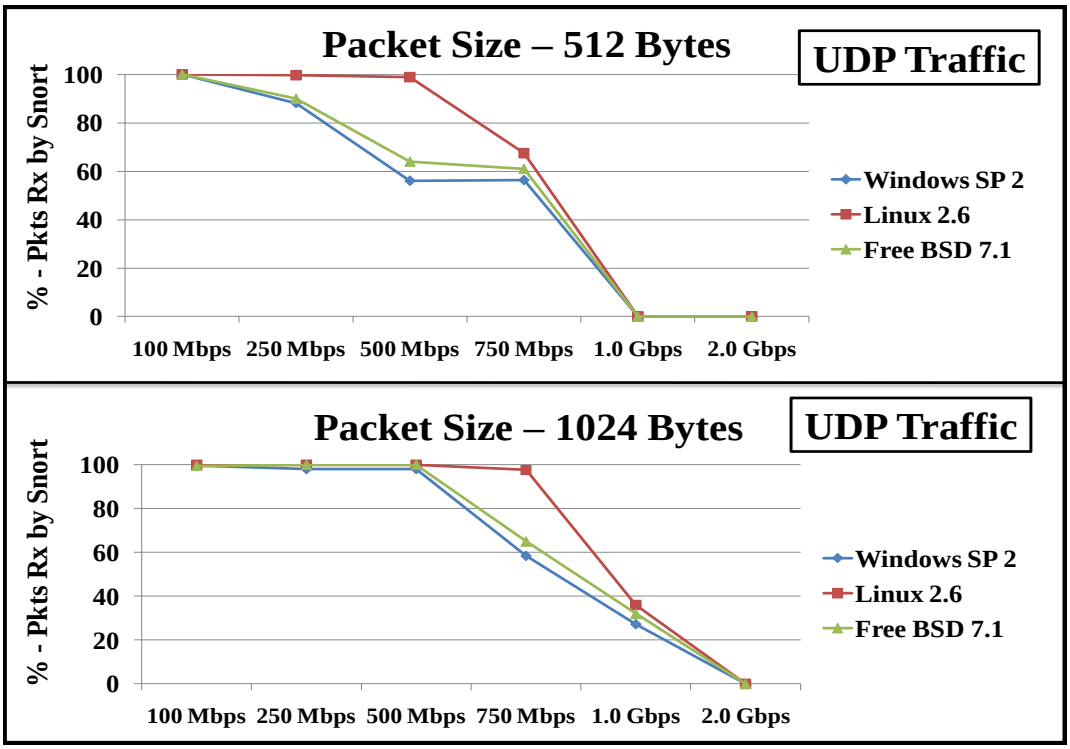

Figure 3.15 Snort packets received (%) – UDP traffic (512 Bytes & 1024 Bytes)

iii. Snort response for packet sizes of 1460 Bytes

- Linux shows significantly better performance for packet sizes of 1460 Bytes for traffic loads up to 1.0 Gbps. However, the system found non-responsive at 2.0 Gbps of loading, as shown in Figure 3.16.
- Windows also showed good performance up to 750 Mbps of loading. The system was found non-responsive at 2.0 Gbps traffic loads, as shown in Figure 3.16.
- Free BSD responded a bit better than Windows. The system was found non-responsive at 2.0 GB traffic loads, as shown in Figure 3.16.

3.8.2.2 TCP traffic

The results of 512 Byte packet sizes have been included in this section due to lack of space. The results have been accumulated on the basis of successful connections (50, 100 and 200 respectively). Packets received at the host platform/OS are shown in Table 3.7.

i. Snort response for 50 connections – 512 Bytes

- Linux exhibits quite good performance up to 750 Mbps of loading; however, its performance declined at higher traffic loads, as shown in Figure 3.16.
- Windows was acceptable up to 250 Mbps of loading but its performance was reduced for higher traffic loads, as shown in Figure 3.16.
- Free BSD performed a bit better than Windows, as shown in Figure 3.16.

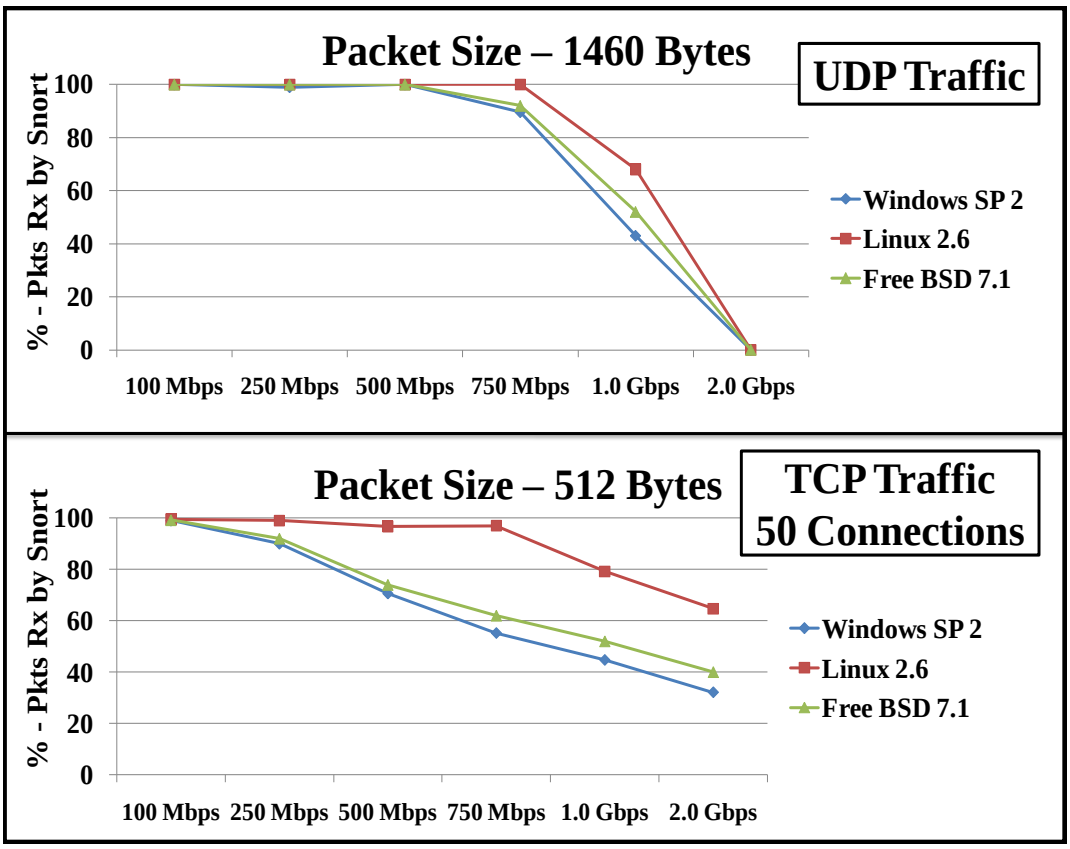

Figure 3.16 Snort packets Rx (%) – UDP (1460 Bytes) and TCP (50 connections).

ii. Snort response for 100/200 connections – 512 Bytes

- Linux exhibited quite good performance up to 250 Mbps of loading with minimum packet loss. However, its response linearly declined for higher traffic loads, as shown in Figure 3.17.
- Windows also exhibited a similar performance level up to 250 Mbps loading levels, but its performance declined for higher traffic loads, as shown in Figure 3.17.
- Free BSD performs a bit better than Windows, as shown in Figure 3.17.

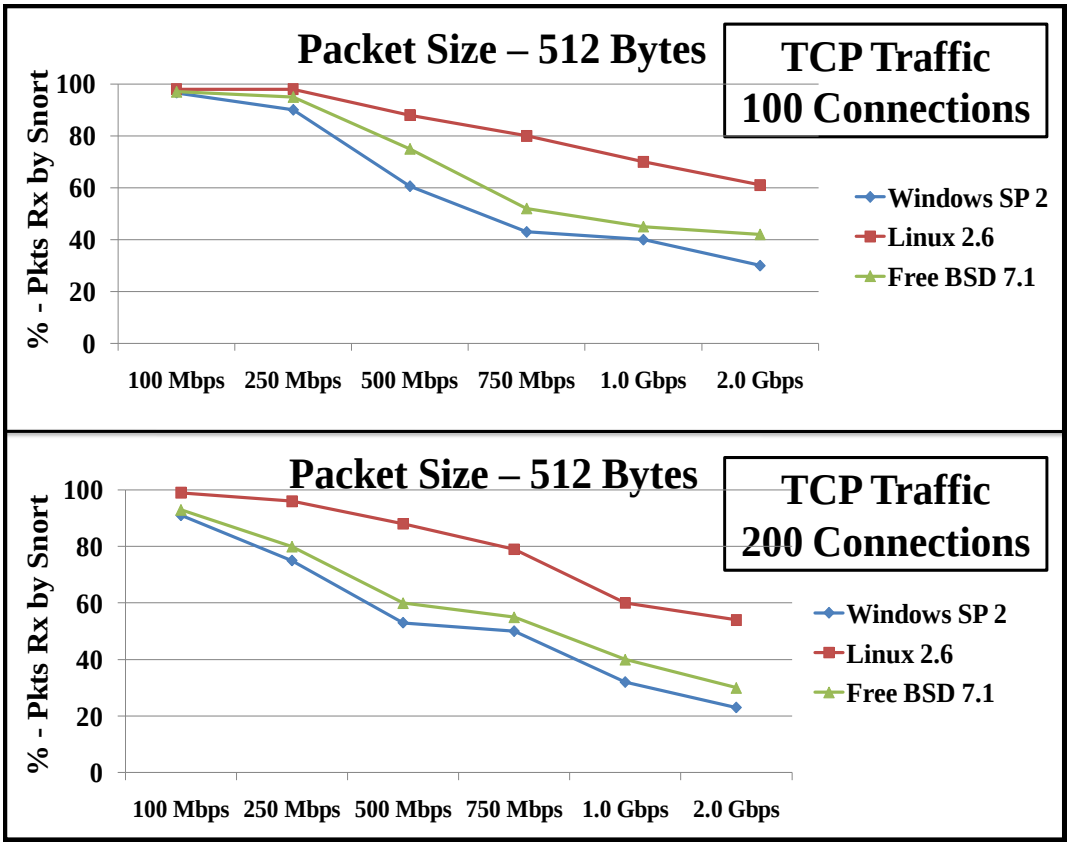

Figure 3.17 Snort packets received (%) – TCP Traffic (100 & 200 connections).

3.9 Analysis

3.9.1 Test-bench 1

As expected, Snort's performance was found to be dependent on its supporting hardware components (CPU, memory, NIC etc.). In the virtual scenarios, Snort was found to be less accurate for all categories of background traffic. Conversely, the performance of Snort improved when run natively on its host machine by utilizing all of the available hardware resources.

Resource constraints in the virtual machine have affected the overall performance of Snort, resulting in a high number of packets dropped and a reduction of alerts logged. The statistics for percentages of dropped packets are shown in Figure 3.18.

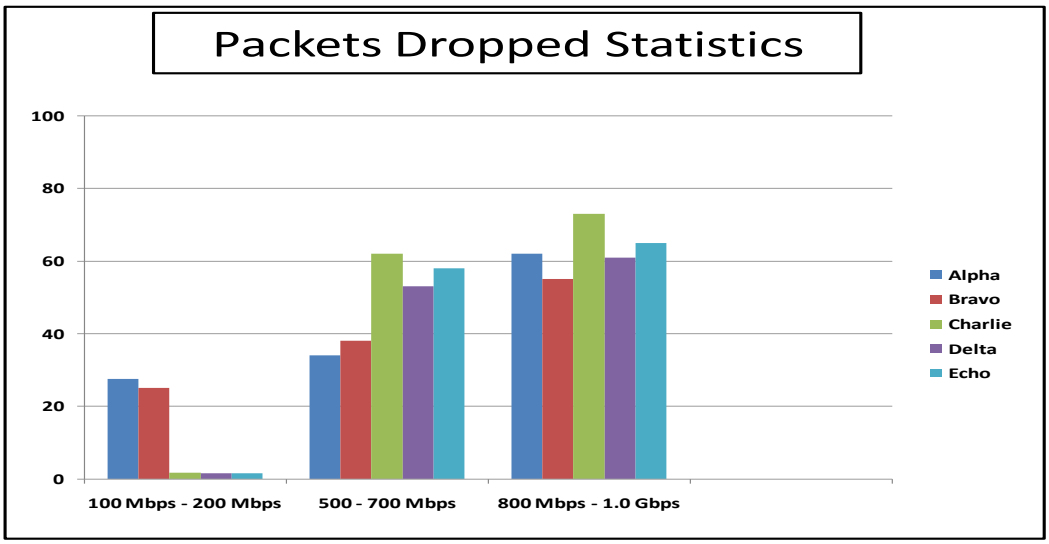

Figure 3.18 Packets dropped.

- Background traffic plays a significant role in the performance of Snort. The higher the traffic, the lower Snort's performance. The impact of background traffic can be ascertained by analyzing the statistics of alerts generated in different categories, as shown in Figure 3.19.

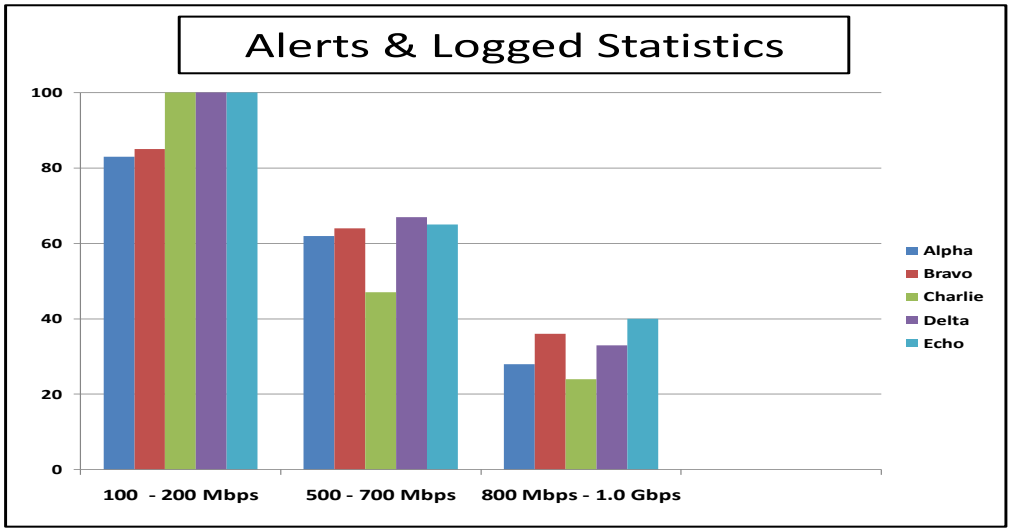

Figure 3.19 Alerts and logs (success rate).

- Traffic within the range of 100–400 Mbps has no significant impact on Snort's performance when run natively on host machines. However, its performance declines in a virtual setup. Snort was found to be accurate in all scenarios.

- A slight increase in background traffic, in the range of 500–700 Mbps, causes deterioration in Snort's performance. This degradation is approximately the same in all scenarios.
- With high background traffic levels, ranging from 800 Mbps–1.0 Gbps, Snort starts bleeding. The number of alerts and log entries suffers significant reduction, thus identifying an evident limitation in Snort’s detection capability.
- In general, Snort was found to be inaccurate when handling traffic levels above 500 Mbps. There was also a significant performance decline when the traffic load exceeded 500 Mbps.
- Snort was found to be more effective in the configuration where both attacker and host are on the same OS.
- Snort's performance is significantly reduced in the 1.0 Gbps scenarios.
- System performance in relation to packet capture capabilities was also found to be dependent on CPU usage. The higher the CPU usage, the lower the number of packets captured for analysis by the Snort application. Packets received at the virtual platform for evaluation by Snort are significantly less than the packets captured at the host platform. However, lower amounts of packets received by virtual platforms result in improved packet analysis statistics by Snort. For example, in the Windows virtual platform, Snort analyzed 38% of the total packets received at system level, whereas in the host Windows configuration, this value was reduced to 27%. The better packet analysis percentage produced by the virtual platform is due to the fact that Snort analyzed a considerably lower amount of packets, whereas the packets captured for analysis at host level were significantly more. Thus, it can by no means be concluded that the virtual platform performed better than the fully resourced host.

- The performance of Snort on a Linux platform was observed to be comparatively better than that of Windows. The results shown in Figure 3.20 are based on the scenarios in which the Snort and attacker are on well-resourced host machines.

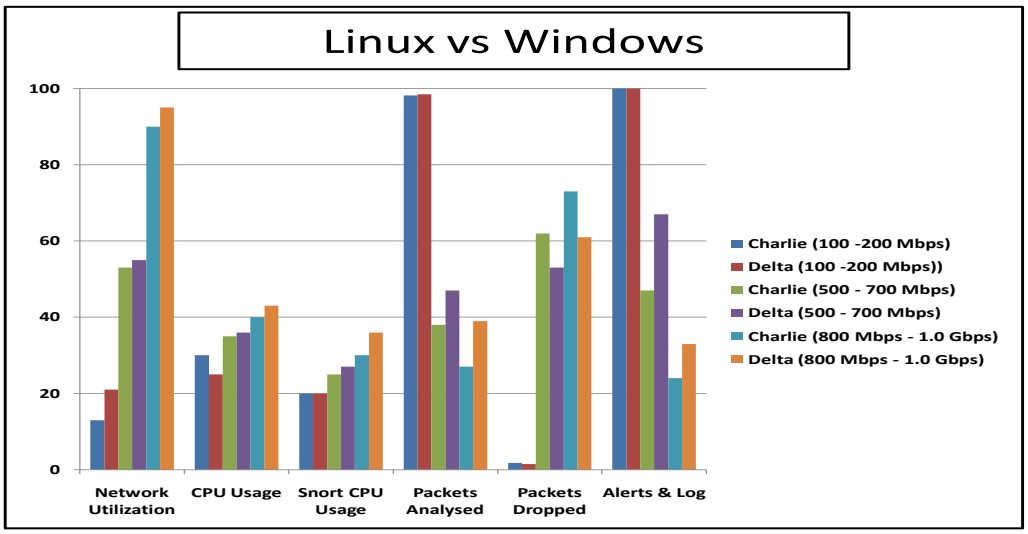

Figure 3.20 Comparison – Snort on Linux and Win.

3.9.2 Test-bench 2

The shaded cells in Table 3.4 indicate the case of the I/O disk bottleneck, when the queue for I/O reading and writing exceeds an acceptable limit and the hosting machine is no longer able to process all the traffic (as discussed in detail below). The overall assessment of system performance indicates following:

- Snort running on Free BSD has achieved the greatest performance in comparison to other OSs for all traffic volumes and packet sizes.
- Windows and Linux showed quite similar performances in all scenarios.
- Small sizes of UDP packets are computationally expensive and the performance of Snort declines in proportion to the increase in traffic bandwidth.

- Considering 1024 Bytes as an average packet size for normal real-life traffic, the raw processing rate of the Snort application showed acceptable performance up to a bandwidth of 750 Mbps for all OSs and 1.0 Gbps for Free BSD.
- The CPU and memory usage of the system for packet sizes of 1024 Bytes (UDP traffic) have been recorded, as shown in Figure 3.21. It has been observed that more than 60% of the hardware strength is available for traffic ranging from 100 Mbps to 2.0 Gbps.

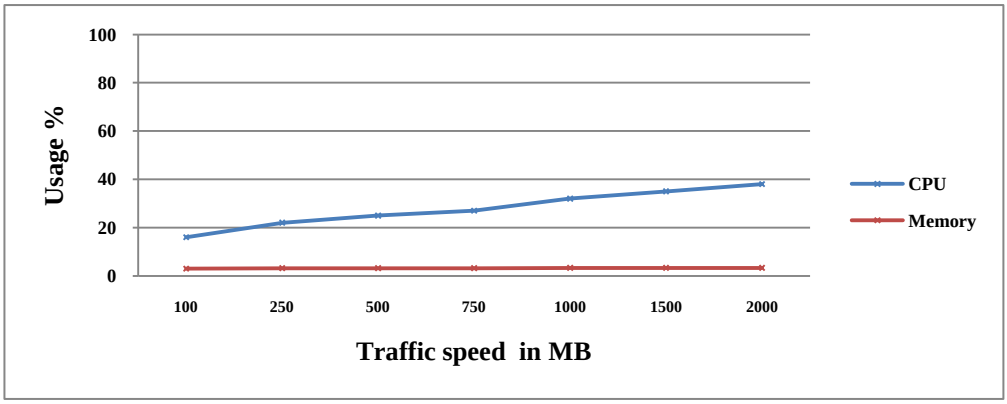

Figure 3.21 CPU and memory usage.

3.9.3 Test-bench 3

We have identified two basic factors that contribute to the packet-drop limitation in virtual platforms running NIDS in high-speed environments.

OS and application incompatibility

The results have identified different packet capture performance levels by the respective OS platforms. The packets received by virtual platforms are actually the packets received by the Snort application. Overall Linux performed quite well in comparison to Windows and Free BSD for both UDP and TCP traffic. The results lead to the following conclusions:

i. UDP Traffic

- All platforms respond well for packet sizes greater than 512 Bytes.
- For packet sizes of 128 Bytes and 256 Bytes, Linux performs significantly better than others; however its performance declines above 250 Mbps loading. Windows and Free BSD performed well for 128 Bytes at 100 Mbps trffic-load only.
- All OS platforms hanged at packet sizes of 128 Bytes and 256 Bytes above 500 Mbps of traffic-load.
- There were practically no measurable results from all the platforms at 2.0 Gbps loading for all packet sizes.
- The overall performance standing measured was Linux, followed by Free BSD, with Windows in last position.

ii. TCP Traffic

- The systems remain alive for all packet sizes and number of connections for traffic-loads upto 2.0 Gbps.
- The performance of the systems linearly declined in response to increases in the number of connections and traffic-load.
- Linux outperforms Windows and Free BSD in all the tested scenarios.

iii. Evaluating OS packet handling competency

In order to reach a definite conclusion concerning OS incompatibility as regards the virtualization of NIDS in high-speed networks environments, the research has been extended to conduct some additional tests. These tests comprised of three virtual machines built on the same OS platform (Free BSD). The Snort application was activated on all platforms and similar tests were conducted as described in section 3.8.1.

In the first scenario, with Free BSD configured on three parallel virtual platforms similar performance metrics were observed. As such the performance of Free BSD was found to be quite similar to the previously executed test-bench scenario and only a small amount of variation was observed. In the second two-machine scenario, an improvement in performance was observed; however performance levels declined at higher traffic-loads. Due to a paucity of space the results of 512 Bytes of packet size for UDP Traffic have been only included as shown in Figure 3.22. The graph shows the average performance of systems in each scenario.

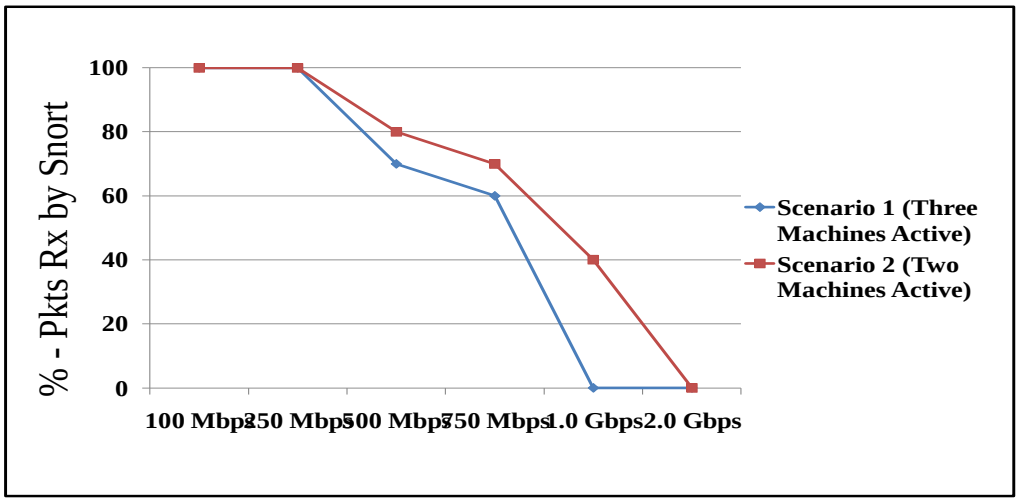

Figure.3.22 Snort Packet Received (%) – Free BSD on Three/ Two virtual platforms

The performance of Free BSD in the two scenarios has identified a direct link between packet capturing ability of the system and the use of hardware resource sharing. The results shows that two platforms perform significantly well in comparison to the use of three virtual machines. Thus, it can be concluded that the packet capturing performance for NIDS when run as multiple virtual instances is limited due to the impact of hardware resource sharing and there is no direct relationship to OS itself. Similar tests were also conducted on Linux and Windows platforms; due to space restrictions the results have not been included. Both platforms behaved in a similar pattern as that of Free BSD thus confirming the drawn conclusion.

Hardware incompatibility in virtualization

The dynamics of virtualization requires the host OS and the virtual machine software (VMware Server) to be stored in the physical memory (RAM) of the host machine. The virtual machines (Windows XP SP 2, Linux 2.6 and Free BSD 7.0) running on a VMware Server have been respectively allocated virtual RAM and disk space on the physical hard drive of the host machine. The processes/applications running on the virtual machines use these simulated virtual RAMs and hard disks for the various operations shown in Figure 3.23.

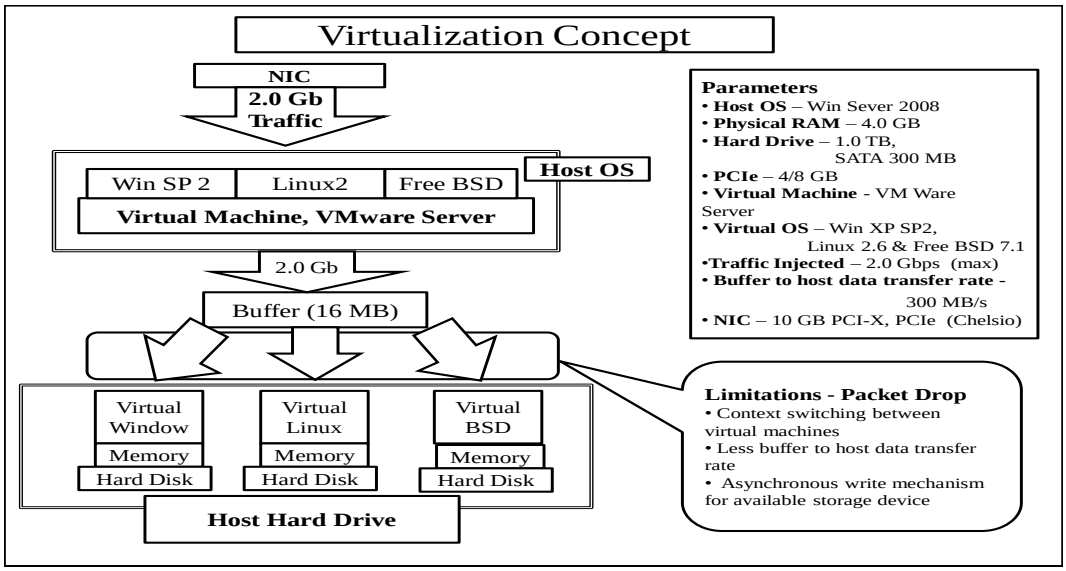

Figure.3.23 Virtualization concept.

Our test-bench has multiple instances of Snort and packet-capture libraries running on different virtual platforms each with a different OS. The packets captured by each virtual machine are less than the packets received by the NIC, thus identifying packet loss. The basic cause of packet loss at each OS, apart from the losses incurred by Snort during evaluation, is the bottleneck caused by a low disk data transfer rate. The disk I/O statistics as shown in Figure 3.24 reflect the hardware limitations in handling multiple read/write operations. At 300 MB/sec of traffic load, the disk I/O capacity touches 100%, thus its performance at higher loads can be easily ascertained.

The memory and storage for each virtual machine has actually been allocated on the physical storage resources (i.e. hard disk) of the host machine. Packets received by the NIC without any loss are transferred to the hard-disk buffer at the PCI rate (4/8 Gbps). From this buffer, these packets are required to be written to the disk at the buffer-to-host transfer rate of 300 MB/sec (SATA Hard Drive) [157]; thus a huge gap between the disk-transfer rate and the incoming traffic load exists. In addition, when traffic is fed to all virtual machines simultaneously (in parallel mode), the disk is physically only able to write to one location at a time. Thus any disk-write instance to a virtual machine will cause packet drops on another. There are also some additional packet losses due to context switching within the hard disk.

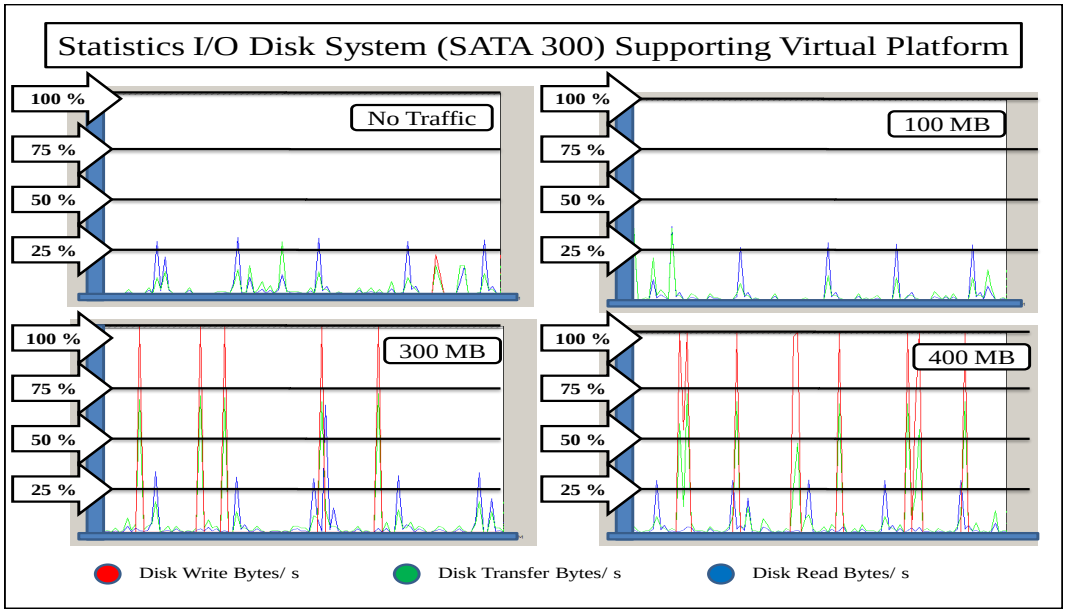

Figure.3.24 Statistics the I/O system (SATA 300) hard drive.

In order to augment our analytical stance showing that hardware is one of the major bottlenecks for the efficacy of the virtualization concept for NIDS in high-speed networks, the disk queue length counter has been utilized as shown in Figure 3.25. In normal circumstances, the average disk queue length should be three or less (its ideal value) [158]. However, in our test network it is observed to be always greater than the

ideal value for the traffic ranges measured at 2.0 Gbps [159].

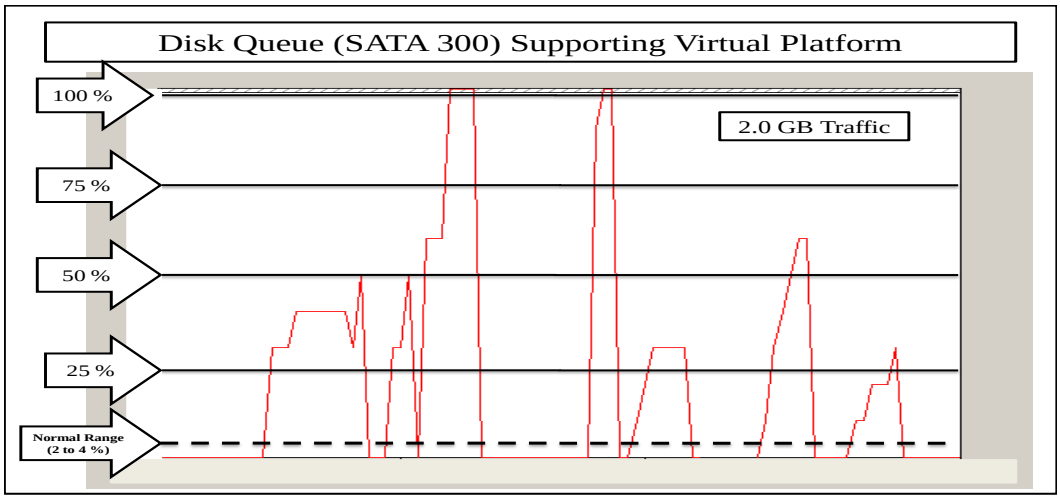

Figure.3.25 Disk queue (SATA 300) hard drive.

3.10 Discussion of Snort performance

3.10.1 Packet processing

Network IDSs and other network monitoring systems are packet-based systems that require packet acquisition from the network wire. A good packet capturing response by the NIDS towards variant traffic reduces the probability of a system becoming compromised. Factors that affect the packet capturing performance of a NIDS in a Gigabit Ethernet environment include i) host configuration (hardware and software) parameters and ii) application-specific parameters (NIDS).

Processing packets can result in a bottleneck for such systems, particularly with high-speed traffic. When these systems become unable to process incoming packets inline, the packets are dropped in order to release system resources (e.g. buffer). The serial packet processing paradigm of Snort implies that a single packet can be processed at a time. The other packets are queued in a buffer, and if the waiting time exceeds a threshold, Snort starts dropping packets for performance reasons. If Snort runs in passive mode, it causes attacks to be missed and violates the coverage requirement.

Furthermore, Snort performs a stateful analysis for packet processing (e.g. TCP reassembly), and if a packet is dropped, this impacts resource utilisation (e.g. CPU and memory).

3.10.2 Multi-core processing

Multi-processing has been introduced using a combination of two or more cores (processors) integrated into a single chip (integrated circuit) to improve performance. However, most legacy applications do not utilise this higher capability because they have been developed for a single processor even if they are run on multi-core systems. The core of Snort, for example, has been developed based on a uniprocessor architecture [24]. In order to exploit current optimization, these applications have to be redesigned to support multithreading. A packet-processing mechanism is one of the critical processes that needs to be enhanced with the existence of multi-core processors. [143] proposed pipelining and flow-pining approaches to improve packet processing in Snort by exploiting multi-core processors. He has shown that running Snort on a multi-core processor does not add any improvement in performance. Modifications on Snort have been implemented to allow multithreading mechanisms. The *parallelism* concept has been employed to spread Snort functions over four cores, which achieves considerable enhancement. [144] also presented strategies for parallel packet processing on multi-core systems by making Snort multithreaded. Several methods have been implemented by separating threads (e.g. a thread is allocated for packet processing and another one for event handling). However, this mechanism has an impact on CPU cache performance and imposes an extra overhead.

3.10.3 PCI bus and disk input/ output (I/O) operations

PCI bus architecture directly influences the operational efficiency of memory and storage devices. Current system architectures identify two bottlenecks in packet-

capturing accuracy: bus traffic load and disk throughput. When writing to a disk, packet capture libraries pass data to the system bus twice, once from the network interface card to memory, and a second time from memory to disk. Thus the actual traffic load available to the PCI bus is half that of the traffic load [145]. Data-intensive applications and the huge amounts of data required to be stored in enterprise networks demand highly efficient I/O operations to avoid performance bottlenecks. The invention of multi-core processors has enhanced the capability of systems to support multiple virtual machines, yet system performance in relation to disk I/O operations remains limited.

3.10.4 Packet loss in NIDSs

Packet loss in high-speed networks is one of the fundamental problems in the implementation of NIDSs. Effectiveness of intrusion detection relies on analysis of the received packets and any loss results in attacks missed. Numerous efforts have been made to address the issues relating to packet loss in high-speed networks. A number of techniques focus on securing a balance between the detection capacity of the system and the input traffic. A few substantially competent techniques make use of load-balancing concepts. These involve the use of a traffic-splitting mechanism where input traffic is distributed across the set of detection engines for evaluation and filtering to block the traffic destined for unpublished ports. [13] explored a parallel architecture to increase the system capacity by splitting traffic into manageable sized slices. The parallel architecture for stateful detection described in [146] also bases its logic on splitting the traffic and distributing it to detection sensors in a round-robin fashion.

However, attacks can be also missed due to the nonexistence of related signatures. It has been observed early in this chapter the relationship between the percentage of packet losses and missed attacks. Hence, a different approach has been adopted to deal with missed attacks in a generic approach. We base our mechanism on the correlation

concept to obtain a global security perspective instead of avoidance strategies. This is based on assumptions that real attack attempts in typical scenarios consist of coherent stages. A framework for alert correlation will be introduced in Chapter 4 to reduce the impact of packet loss and missed attacks.

3.11 Conclusion

This chapter has focused on ways of determining the efficacy of the widely deployed open-source NIDSs, namely Snort, in high-speed network environments. The current development in hardware technologies has opened broad prospects for legacy applications, particularly software-based ones deployed at network edges. Multi-core systems are available and widely used to offer intensive computational opportunities.

The test scenarios employed involve the evaluation of the application under different traffic conditions, and observing the response of the system to known attack signatures. The results obtained have shown a number of significant limitations to Snort, on both host and virtual configurations. We have confirmed that the underlying host hardware plays a prominent role in determining overall system performance. We have further shown that performance is further degraded as the number of virtual instances of NIDSs is increased, irrespective of the virtual OS used.

This hardware dependency is exacerbated when running Snort as a virtual machine, and it is to be anticipated that running a large number of Snort instances would lead to major degradations in performance and detection levels. In general, any limitations in system configuration would result in poor performance of the NIDS. The results obtained have shown a number of significant limitations in the use of virtual NIDSs, where both packet-handling and processing capabilities at different traffic loads were used as the primary criteria for defining system performance. Furthermore, It has been

demonstrated a number of significant differences in the performance characteristics of the three different virtual OS environments in which Snort was run.

In the pursuit of our objective, the performance of Snort has been analyzed under realistic network conditions in contrast to simulated testing environments. The results obtained identify a strong dependency from Snort on the host-machine configuration. It can be ascertained that Snort is not suitable for all network implementations with high volumes of traffic, e.g. more than 750 Mbps.

It has also been identified the impact of packet loss caused by performance degradation upon the overall effectiveness of NIDSs. We intend to introduce a dual solution for both packet loss and missed attacks using alert correlation, as will be shown in the following chapters.

CHAPTER 4: A REASONING FRAMEWORK FOR ALERT CORRELATION

4.1 Introduction

In an intrusion detection context, none of the main detection approaches (signature-based and anomaly-based) are fully satisfactory. False positives (detected non-attacks) and false negatives (non-detected attacks) are the major limitations of such systems. The generated alerts are elementary and in huge numbers. In addition, It has been identified in Chapter 3 that even though the attack signature is defined in the attack database, it can be missed in high-speed environments. This has made the issue more complicated, reducing the attack detection rate. A promising approach is to incorporate a collection of security detection systems to increase the detection coverage whilst at the same time suppressing the volume of false positives. Hence, alert correlation techniques are used to provide a complementary analysis to link elementary alerts and provide a more global intrusion view. On the other hand, alerts generated by a single IDS also overwhelm the administrator and contain a high percentage of insignificant or irrelevant information. It has been widely recognised that real cyber attacks consists of phases that are temporally ordered and logically connected. In this thesis, the focus is on the correlation function of the alerts sourced from the same IDS. The objective is to discover the logical relationships between atomic alerts potentially incorporated in multi-stage attacks. An alert correlation and aggregation framework is presented based on *requires/provides* model [121].

This chapter explains the fundamental concepts of the proposed alert correlation framework. The correlation process is essentially modularized based on an extension of the properties and characteristics of the *requires/provides* model [121]. The description

of the knowledge base modelling is based on the capability concept to abstract alerts sets to their pre- and post-conditions. Capability conditions formalization is also explained, which is mainly based on a proposed hierarchical abstraction of attack classes. Algorithms of alert correlation, alert aggregation and graph reduction are presented. And finally, the prediction of undetected attack action is discussed.

4.2 Multi-stage Attack Recognition System (MARS) framework

The MARS framework is a logical framework supported by various components for alert correlation, aggregation, reduction and multi-stage attack recognition, as shown in Figure 4.1. Despite the differences between alert correlation approaches, they require some common modelling. A knowledge-base that contains attack characteristics is either abstracted or using actual attack details. Information acquisition for a knowledge base is based on the model employed (e.g. expert systems, artificial intelligence). The main drawback of the previous approaches is that they do not provide knowledge representation in a systematic way. For instance, *requires/provides* is a general alarm management model that has been used widely in the alert correlation field, but most of the proposed paradigms are based on ad hoc methods of knowledge representation. In our framework, knowledge elements are designed using a formal knowledge formalization exploiting available information provided by IDSs, vulnerability scanners and environment configurations. It also allows interactive communication between the administrator and the core system engine. Generated events reflecting the detected security situation are produced after a series of processing functions to reduce the data size. The implementation of the MARS framework will be discussed in Chapter 5. In this chapter, the underlying principles of the proposed framework are introduced.

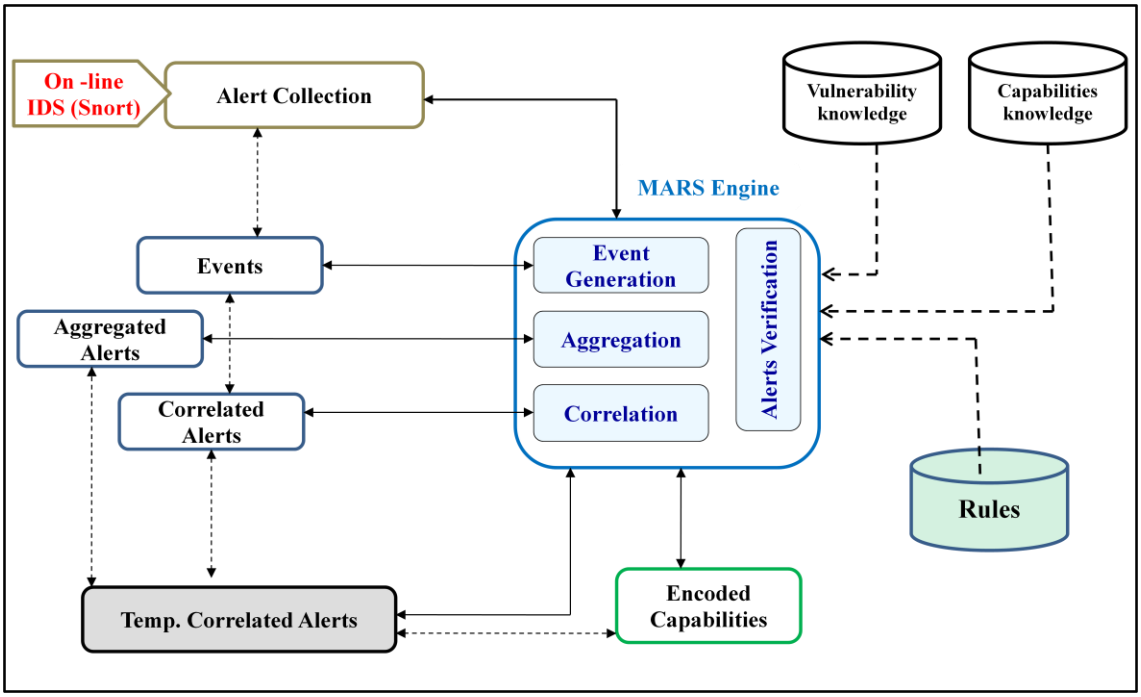

Figure 4.1 Multi-stage Attack Recognition System (MARS) framework.

Figure 4.1 gives a graphical representation of the framework components that implemented in MARS system. The first task is performed on all received alerts from the IDS sensor e.g. Snort. *Alert Collection* contains normalized alerts presented in a standardized format that are understood by all correlation components. Also, a pre-processing function is carried out to normalize all required alert attributes such as time stamp, source, and destination addresses. The final results of this process are stored in *Alert Collection* which represents the main data input for the MARS engine. MARS engine consists of four components: 1) *Alert Verification* 2) *Correlation* 3) *Aggregation* and 4) *Event generation.* The task of *Alert Verification* component is to take a single alert and determine the success of the attack that corresponds to this alert. Failed attack should be assigned as a low level of importance. However, these failed attacks are not ignored and saved in the database which can be used as evidence to support other correlation instances. The *Aggregation* component is responsible for combining a series of alerts that refer to attacks related to the same activity. IDS sensor produces number of

alerts corresponding to the same attack which are conducted at the same time. Similar alerts are aggregated and a representative alert is assigned based on a temporal relationship. These aggregated alerts are saved in the aggregation collection and are used to generate multi-stage attack events. The main task of the *Correlation* component is identifying the logical connection between received alerts based on the used correlation algorithm. If any link between two alerts is recognized, they are correlated and stored in a temporary collection and then transferred to the correlation collection after performing the aggregation process. The task of the *Event Generation* component is identifying and constructing multi-stage attack patterns which are composed of a sequence of individual alerts. A new event is generated if at least two alerts are correlated and then the generated events are stored in the *Events* collection.

Two knowledge bases are used by MARS engine to support the correlation process: 1) Capabilities Knowledge base and 2) Vulnerabilities knowledge base. The capabilities database contains modelled attacks and the relationships between different attacks based on pre and post conditions of each modelled attack. Snort signatures are used in the current implementation and this can be extended to include attack definitions from other sources. Vulnerabilities database contains network and host configuration of the protected system in addition to the detected vulnerability information by the available scanner.

The initial task executed by the MARS engine is obtaining alerts from the alert collection and then creating encoded capabilities corresponding to each alert. Alerts attributes and the information supplied by the used capabilities knowledge base are used to build the encoded capabilities collection. Thus, the encoded data is utilized to produce the initial correlation information and then it is stored in the *Temporary Correlated Alerts* collection. This collection contains atomic logical connections

between alerts which are consequently aggregated to obtain the aggregated collection. The generated events (Multi-stage attack instances) are constructed based on the aggregated alerts in order to minimize the resulting graph.

4.3 *Requires/provides* model

This model is a general attack model that has been proposed by [121] and is inspired from network management systems to deal with network faults. A cyber attack is described according to two components: 1) capabilities, and 2) concepts. The idea behind this model is that multi-stage intrusions consist of a sequence of steps performed by an attacker, and that the later steps are prepared by the early ones. The target system information collected from scanning or port mapping are advantages acquired and used in order to choose which exploit can be successful. Attacks are modelled in terms of abstract concepts and each concept requires certain capabilities (conditions) to occur and provides others to be used by another concept. Capabilities are defined as general descriptions of the conditions required or provided by each stage of the intrusion i.e. the system state that must be satisfied in order to launch an attack. For instance, a successful Trojan injection requires particular services to be running in the target system and the presence of certain vulnerabilities.

Formally, capabilities are a higher level of intrusion abstraction that specifies the system state after each attack attempt. The attacker uses the capabilities acquired through some of its early actions to generate certain new capabilities. The system state is incorporated in attack scenarios if instances of concepts have matched "*required*" and "*provided*" conditions.

The capability model proposed by[160] is also based on a *requires/provides* model for logical alert correlation, though the authors used different properties of capabilities. An

attack model was presented to build blocks of capabilities in a multi-layer fashion and with more expressive definition. [35, 49] have employed a *requires/provides* model using the concept of predicates, which are similar to capabilities.

Both models mentioned above are reasoning models that aim to discover the causal relationships between elementary alerts. Attacker states are abstracted to describe the gained privileges and what level of access is obtained. Moreover, the system states are modelled into a higher level of abstraction to specify the impact of the attack. Relationships between these states are defined to generate rules that determine the dependency between alerts.

Requires/provides model has been selected because it fits our purpose to correlate alerts in the same intrusion. It has some advantages over other models:

1- Ability to uncover the causal relationships between alerts and it is not restricted to known attack scenarios.

2- Ability to characterize complex scenarios or to generalize to unknown attacks.

3- Attack is represented as a set of capabilities that provides support for the abstract attack concepts.

4- Flexibility and extensibility as the abstract attack concept are defined locally.

5- It does not require a priori knowledge of a particular scenario.

6- Numerous attacks can be described implicitly and unknown attack can be defined by generalisation.

Our approach is a variation of the *requires/provides* model, but differs in the following aspects:

- Different definitions for capabilities and concepts are employed to overcome the limitations expressed in other approaches. The work in [121] used a very detailed

specification language called JIGSAW to describe attack scenarios. A complete satisfaction of "*required*" and "*provided*" conditions is necessary to correlate two alerts, which will fail in case of broken scenarios. However, the authors in [35] have adopted a partial satisfaction technique which is also implemented into our framework. The main concern with their approach is the high rate of false positives, and the possibility of a huge graph being created. We have managed to overcome this limitation by using certain techniques: hierarchical multi-layer capabilities, accumulated aggregation, alert verification and alert maintenance.

- A near real-time processing approach for correlation, aggregation and event generation. The security officer can monitor the attack progress which is displayed as an intrusion graph. An event is triggered once at the minimum of two alerts being correlated, and any additional related alert based on its attributes will join the same event.
- Online and offline graph reduction algorithms during the correlation process in addition to alert aggregation in order to provide a smaller manageable graph.
- We have modelled IDS signatures as abstracted attack concepts instead of defining new concepts locally. In *requires/provides* models, IDS signatures are considered complementary external concepts.
- Separation of the concepts and their capabilities from other dynamic information. Two different types of capabilities have been used: internal and external. The first type denotes abstract attack modelling consisting of IDS signatures and associated capabilities. The second type refers to dynamic details, including system configuration, services and vulnerabilities. This provides more flexibility to the model whilst at the same time allowing utilization of other knowledge resources.

- A capability modelling has been made using a hierarchical methodology based on attack classes and inheritance between these classes.

Our approach is based on the assumption that the attack scenario consists of a sequence of related actions and that early stages can incorporate later ones. The link between these stages is determined using five factors:

1- Temporal relationships (e.g. alert timestamps).

2- Spatial relationships (e.g. source IP addresses, destination IP addresses and port numbers).

3- Pre- and post-conditions of each attack.

4- Vulnerability assessment of the target system.

5- Target system configuration.

Capabilities are formalized in term of pre- and post-conditions by grouping conditions that share similar characteristics into a broad definition. Knowledge about elementary alerts is mapped to instantiate the attacker and the system states according to their temporal characteristics:

- *Pre-conditions*: are logical capabilities that characterize the system state to be satisfied in order to launch an attack. These capabilities are derived from the attack description. A hierarchical approach is adopted based on an attack classification to provide coarse-grained definitions of different alerts related to the same behaviour.

- *Post-conditions*: are logical capabilities that characterize the system state after the attack succeeds. In other words, specifications of the effects of intrusions on the system, such as the knowledge gained and the access level of the attacker.

Moreover, attack classification incorporates the definitions of these capabilities in a hierarchical manner.

To formulize the capability sets as pre- and post-conditions of higher quality, certain requirements must be satisfied:

1- Capabilities must be expressive in order to achieve a true logical relationship.
2- Avoidance of ambiguity in defining capabilities.
3- Use of multi-layers of abstraction to achieve scalability.
4- Reduction of the number of elements in the capability sets without affecting attack coverage.
5- Inference rules should be separated from the capability set.
6- The set should also be constant and independent of variable information such as vulnerability and system-configuration knowledge.

Hence, capabilities are formulized based on two criteria:

1) Level of abstraction

1- Generic capabilities which illustrate a broad aspect of a certain attacks, such as access, local access and remote access.
2- Capabilities which illustrate a lower level of attack abstraction, but not a specific one, such as server buffer overflow or client upload file.
3- Specific capabilities for each single alert in IDSs, such as TFTP Get.

2) Properties of the system and the attacker state

1- Access level of the attacker (remote, local, user or administrator).
2- Impact of the intrusion upon the victim machine, such as DOS and implementation of the system commands.
3- Knowledge gained by the attacker, such as disclosure of host or of service.

The elements in the two criteria above are mutually inclusive; for instance, *disclosure of host* is considered as a generic capability and at the same time is a system state description. In addition, attack classification, which will be presented in the next section, is also involved in defining capabilities.

Examples: generic capabilities are mainly a description of the intrusion's general objective, such as:

- Disclosure of host
- Disclosure of running service
- Disclosure of port number
- Access
- Read or write files

However, a *buffer overflow* attack is a general attack that can target the server, the Web server and the client, and the required and provided conditions are not the same for each category. The capability *client access attempt* is a specific capability for client attacks, because some attacks are client specific, such as ActiveX attacks. Snort documentation contains a description for each signature, including the attack class type, the affected system, and the impact of the attack. This information is valuable in defining attack capabilities if other sources of intrusion analysis are considered. Appendix I contains an example of a Snort signature description.

4.4 Knowledge-base modelling

Two knowledge bases are used, one for attack concepts and the other for vulnerability details. In the attack knowledge base, IDS signatures (e.g. Snort) are modelled to the attack abstractions and their defined capabilities. The knowledge library specifies the relationship between low-level alerts and the attack abstraction. Thus, a knowledge base

can be considered a broad template and each element can be instantiated to instances of specific conditions. A generalization mechanism has been used to specify a higher level of specification of attack concepts and capabilities.

The proposed model for the attack knowledge base consists of three sets:

1) Capability *C*: This specifies a higher level of abstraction of the "*required*" and "*provided*" conditions of the intrusion model. Intrusion attempts are expressed in terms of a set of "*required*" or "*provided*" conditions, and vulnerability constraints of a given alert where:

- Required conditions *R* are a set of pre-conditions specified in the form of capabilities with variable arguments.

- Provided conditions *P* is a set of post-conditions specified in the form of capabilities with variable arguments.

- Vulnerability *V* is a description of the state of the target host or network with variable arguments.

2) Attack concept *AC* specifies the constructor of a given attack and its related capabilities. "*required*" and "*provided*" conditions for each attack are coded in a language of capabilities.

3) Arguments $[r_1, r_2, \ldots r_i] \rightarrow r$ are a set of associated attributes such as source IP addresses, destination IP addresses and port numbers.

Definition 4.1: Attack concept *AC* is an abstraction of elementary alerts generated by the IDS, defined by a set of arguments, required conditions and provided conditions.

Definition 4.2: An attack instance a_i is defined as a set of instances of attack concept *AC* by substituting the associated values in arguments tuple considering the time constraints (start-time and end-time).

Definition 4.3: Given an attack concept *AC*, the *R(AC), P(AC)* and *V(AC)* sets are the sets of all capabilities C. Given an attack instance *a*, the *R(a), P(a)* and *V(a)* sets are the capabilities by mapping the values to the corresponding arguments in *AC* considering the time constraints.

4.4.1 Attack classification

Several attempts have been made to propose a different attack taxonomy or ontology; however, they are diverse and there is no common methodology for the categorization of security intrusions. The majority of the proposed classifications are entirely based on the analysis of published vulnerabilities. For instance, NIDS vendors such as Snort use attack classes that describe the attacker's methods in exploiting these vulnerabilities. We have obtained our classification based on:

- Vulnerability analysis
- Generalized description of the target system (server, client, Web, etc.)

Elementary alerts generated by NIDS sensors are mapped to generalized descriptions of intrusion in a hierarchical representation. The classification is built in the form of a graph with nodes and edges. The nodes specify the attack class and the edges denote the inheritance relationship between attack classes. The classes are mutually exclusive and each alert belongs to only a single class horizontally, but to different classes vertically based on the inheritance relationship. This structural abstraction mechanism is to minimise redundancy and maximize diversity. Hence, even though some alerts are new and unknown, they can be predicted from the results of situation analyses. If an attack is in progress consisting of certain elementary alerts, these atomic alerts are mapped to a general attack description. For any suspicious or unknown actions not detected by the IDS, the probability of their being related to the detected attack is very high. The level

of the abstraction progresses from general to specific in a top-down design of the classification graph as shown in Figure 4.2.

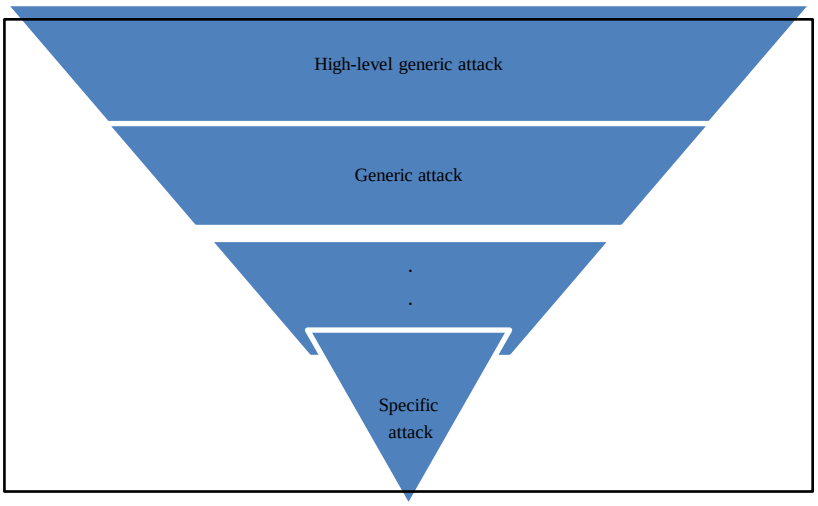

Figure.4.2 Abstraction levels of attack classification.

#	High-level generic attack categories
1	Server attack
2	Client attack
3	Web attack
4	Malicious software activity
5	Suspicious behaviour
6	Policy violation
7	Other

-a-

#	Generic attack categories
1	Buffer overflow
2	Brute
3	Download
4	Information Gathering
5	Implementation of the system commands
6	Landing behaviour
7	Bypass authentication
8	Bypass authorization
9	Upload
10	DOS
10	Cross-site scripting
11	File modification
12	File Inclusion
13	Penetration testing
14	SQL injection
15	Webshell install
16	DDOS client activity
17	Webshell activity
18	Rouge software
19	Trojan
20	Attack response
21	Session Hijacking
22	Scanning

-b-

Figure.4.3 Attack classification.

In Figure 4.3 (a), the high-level generic attack classes are shown and each class can be linked to one or more other generic class in Figure 4.3 (b). For instance, the buffer overflow class can be classified under server, client or Web classes, as this type of attack can target different types of systems. However, some other classes are only categorized as specific system classes, such as DDoS client activity, which is a client-specific attack. Hence, each alert generated by the IDS will be categorized top-down in a hierarchical manner. Figure 4.4 shows three examples of how sub-classes inherit attack features from upper classes and how alerts are classified based on these relationships. In Figure 4.4 (a), the lower class denotes the exact Snort signature *TFTP Get, id:1444*, while this signature is classified as *TFTP buffer overflow*. Similarly, in Figure 4.4 (b), any IDS signature of type of *ACTIVEX attack* can be classified under this class which is in turn classified as a *client buffer overflow*. Figure 4.4 (c) shows that a *stored procedure attack* is described as a Web PHP injection attack. It should be noted that these are only abstract classes and do not denote instances of actual attacks.

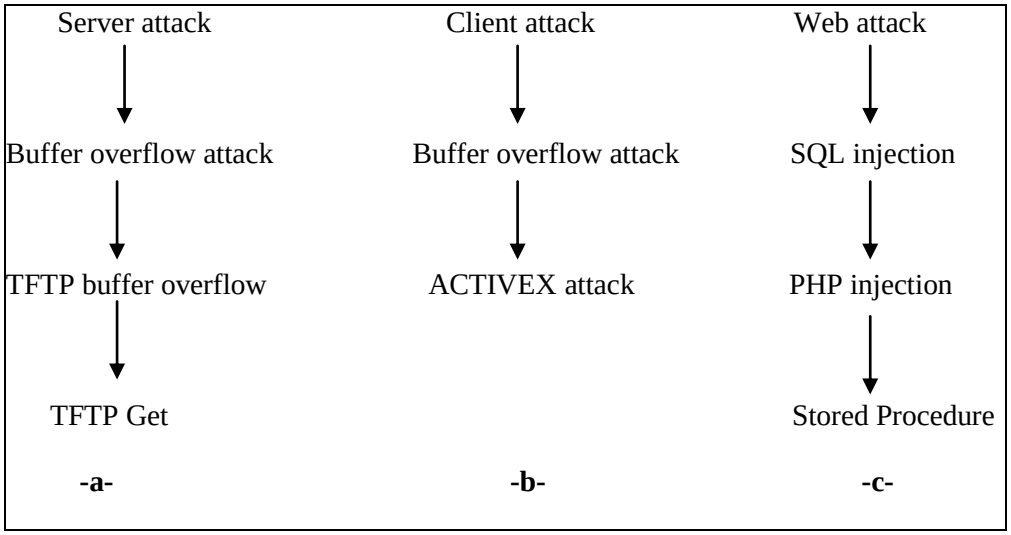

Figure.4.4 Examples of attack class inheritance.

4.4.2 Knowledge-base representation

A capability set consists of all the derived elements of capabilities encoded to integer numbers. All alerts are represented in the form of three sections:

1- IDS signature ID to describe the attack by its elementary alert.

2- Pre-conditions set which consists of n capabilities where n>=0.

3- Post-conditions set which consists of n capabilities where n>=0.

The knowledge library of the alerts and their corresponding capabilities are defined into the form shown below:

sid:$xxxx$;pre:$k_1(n)$;pre:$k_2(n)$;……pre:$k_i(n)$; pos:$l_1(n)$;pos:$l_2(n)$;…..pos:l_j, where $xxxx$ is the signature ID number, pre denotes pre-conditions, pos denotes post-conditions, k is the capability unique number, and n is a variable argument to specify the attack attributes as follow:

1: source IP address

2: source port

3: destination IP address

4: destination port

4.4.3 Alert modelling

IDS alerts are the basic units that represent the occurrence of intrusion as a time series. Essential attack knowledge is derived from signature fields triggered by the IDS in case of any security violation. It should be noted that the alert generated by the IDS is not necessarily connected to a security attack, as sometimes a legitimate activity can cause some alarms. Moreover, the information in the signature does not contain any sign of whether the attack succeeded or not. However, the abstraction of these alerts to capabilities in respect to temporal and spatial details can give a true view of the security perspective.

Each received alert is mapped to its pre- and post-conditions. It is assumed that the alert is generated because some conditions have to be satisfied and that it will cause some

impact on the target system. The relationship between different alerts is identified by matching these conditions, as shown in Figure 4.5.

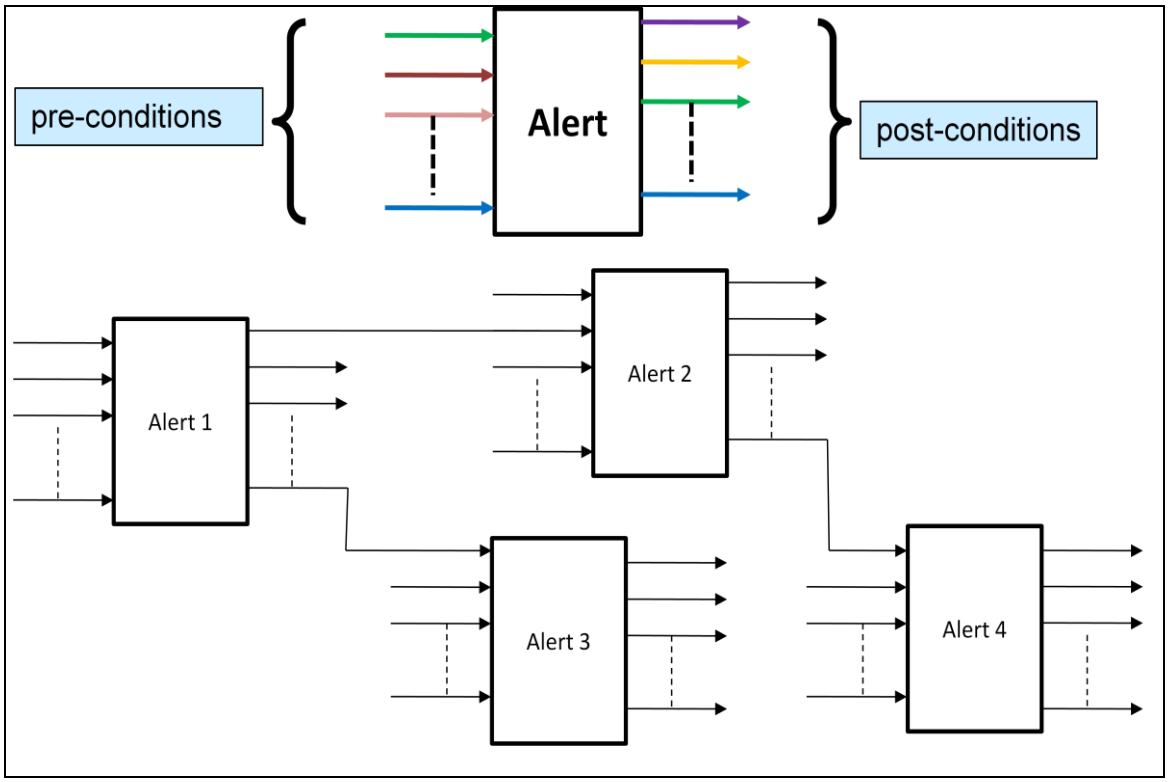

Figure 4.5 Matching of alert pre-and post-conditions in the correlation function.

In Figure 4.5, *Alert 1* has some pre-conditions and post-conditions and one of its post-conditions match the pre-conditions of *Alert 2* and *Alert3*, hence they are correlated. In addition, the temporal order of *Alert 2* and *Alert 3* is taken in account. For example, the following alerts (Snort-generated signatures) are obtained from DARPA LLDDOS.1.0 [161] to clarify the correlation concept considering the following Snort signature:

RPC sadmind UDP PING

This signature is generated as result of attempts to test if the *sadmind* demon is running. A *sadmind RPC* service is used to perform administrative activities remotely. The impact of the signature includes disclosure of the running service and system access attempt:

RPC portmap sadmind request UDP

This signature is generated due to the use of a *portmap GETPORT* request to discover the port number of the RPC service, and consequently which port is used by the *sadmind* service.

RPC sadmind UDP NETMGT_PROC_SERVICE CLIENT_DOMAIN overflow attempt

This signature is generated as a result of an attempt to exploit a buffer overflow to obtain a root access.

RPC sadmind query with root credentials attempt UDP

This signature is generated due to the use of root credentials and is an indication of potential arbitrary command executions with root privilege.

RSERVICES rsh root

This signature is generated due to an attempt to login as a root user using *rsh*, and this is an indication of full control of the attacker.

Table 4.1 Examples of pre- and post-conditions.

#	Signature	Pre-conditions	Post-conditions
1	RPC sadmind UDP PING	Disclosure of host	Disclosure of running service System access
2	RPC portmap sadmind request UDP	Disclosure of host	Disclosure of port number Disclosure of running service System access Remote Access
3	RPC sadmind UDP NETMGT_PROC_SERVICE CLIENT_DOMAIN overflow attempt	Disclosure of host Disclosure of port number Disclosure of running service	System access Remote access Admin access
4	RPC sadmind query with root credentials attempt UDP	Disclosure of host Disclosure of port number Disclosure of running service System access Remote access	Remote access Admin access

From Table 4.1, it can be seen that the signatures have some pre- and post-condition and if a match between these conditions is detected the two alerts are linked as a part of the attack scenario, as shown in Figure 4.6. The two signatures share at least one common capability, *disclosure of running service*, hence they are correlated. It should be noted that the correlation process does not simply consist of matching these capabilities – there are other factors involved, as explained in the rest of this chapter.

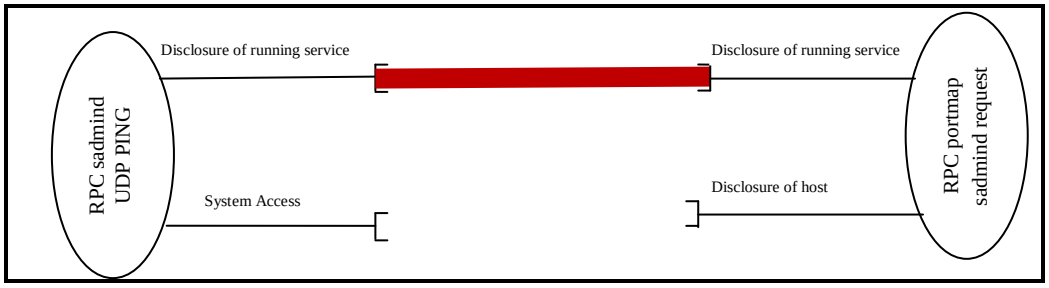

Figure 4.6 Correlation of two alerts.

4.5 Vulnerability modelling

Several efforts have been made to correlate IDS signatures with vulnerability information. The aim is to reduce the false positives, which can be a major drawback of such systems. Moreover, these verification mechanisms are incorporated in the IDS to provide a higher quality of alerts, and hence more confidence. The origin of the problem of false positives is that IDSs have no information about the systems they protect. Therefore they are not certain about the success of the attack, simply because the vulnerabilities of the target system are not available. Two trends have emerged in overcoming the false positives issue in IDS performance:

1- Tuning the IDS based on knowledge of the internal policy of the protected environment to operate with a lower number of signatures [47,48]. Knowledge of network configuration, running services and installed applications is used to disable all the unrelated signatures of the IDS. The advantage of this technique is that the IDS performance is improved significantly. However, some of the information on the activities of the attacker, which may be useful in tracking its behaviour, will be discarded. It should also be noted that real cyber attackers (persistent attackers) try to break into systems using different methods, and these attempts may be not in connection with a particular vulnerability. Moreover, some dangerous attacks in cyber crime do not require any system vulnerability, such as

DDoS. In addition, this approach requires intensive and updated vulnerability assessment.

2- The other trend is not suppressing the IDS detection coverage, but instead aggregating, correlating and verifying the generated alerts in a systematic way [10,29,49]. Summarized data of occurring events are displayed to the security manager according to their priorities and criticalness. If further details are required to support a specific situation, they can be retrieved by request. A repository of collected information is maintained to support the decision of the IDS management system. Vulnerability scanners are the main candidate to supply this type of data in a periodical manner.

In accordance with the nature of the developed correlation systems, which require full description of any activity in the protected environment, the second mechanism is adopted in our research. In the previous sections, the attacks are generalized to obtain a global view of the security situation. This generalization may increase the false positive rate; hence, a suppression technique is needed to reduce the false positive rate without losing any details. This suppression mechanism does not imply any reduction in the IDS coverage, but the consideration of only success attacks.

Snort signatures are supported by two useful fields:

- Vulnerability reference, referring to the major vulnerability standards such as CVE [39], bugtraq [162], and Nessus [128].
- Service to denote a list of the affected services, such as telnet, ftp and MSSQL.

A vulnerability knowledge base is maintained to store the vulnerability situation of each element of the protected network based on the collecting agent (e.g. Nessus). The scanner will also gather the network configuration details such as IP addresses of live

hosts and running services, so manual configuration is not considered. In this respect, vulnerability information is considered as external capabilities.

The scope of vulnerability testing is limited to only investigate the presence of the vulnerability and the affected service. An extension can be carried out to consider the target host response; however, there are performance issues (e.g. communication overheads). Nessus is used to extract the following information, which can be used to support the vulnerability component:

- IP addresses of all hosts connected to the target network.
- Operating systems and their versions.
- Open ports and running services.
- Related vulnerability references (e.g. CVE).

When an alert is received from the IDS, its message contains the vulnerability reference and the affected system. Therefore, a logical formula is obtained by searching the vulnerability knowledge to find any matches, as follow:

- If the reference is found and the associated service is running, then the vulnerability is true with high priority.
- If the reference is found and the associated service is not running, then the vulnerability is true with low priority.
- If the reference is not found, then the vulnerability is unknown.

The complete algorithm of alert verification using vulnerability knowledge is shown in Figure 4.7.

Algorithm :Alert verification

Input: elementary alerts generated by IDS *A*(*IP,SV,VR*)

Host vulnerability information generated by scanner *VN*(*IP,OS,SV,VR*)

Output: Vulnerable host *VH*(*IP,V,P*)

Methods:

// IP: IP address, SV: service, VR: vulnerability, OS operating system

for *i*←*0* to *length*[*VN*]

do

if *A.IP* = *VN*[*i*].*IP* **get** *VN*(*IP,OS,SV,VR*)

in case of

A.VR=VN.VR **and** *A.SV=VN.SV* **then** *VH.V*←*true , VH.P*←*high*

A.VR=VN.VR **and** *A.SV≠VN.SV* **then** *VH.V*←*true ,VH.P*←*low*

A.VR≠VN.VR **then** *VH.V*←*false , VH.P*←*unknown*

Figure 4.7 Alert verification algorithm.

4.6 Alert correlation algorithm

The principle objective of the proposed framework is to identify the causal relationships between a series of attacker actions that are temporally ordered. The concept of alert correlation should not be confused with alert aggregation or alert fusion, as the latter group alerts based on clustering regardless of their temporal relations in some approaches. Alert correlation is the process of identifying a sequence of distinguished alerts that fall in the same generalized attack pattern. Figure 4.8 shows the relationship between alert correlation and alert aggregation. Correlation functions are performed across the x-axis and aggregation functions along the y-axis. In this regard, we do not need to define explicitly the attack scenario, and instead the logical rules are generated using the pre- and post-conditions of each activity. Attributes provided by elementary alerts are used to define instances of alerts. Instances of system conditions are instantiated with time constraints, and correlation rules are created.

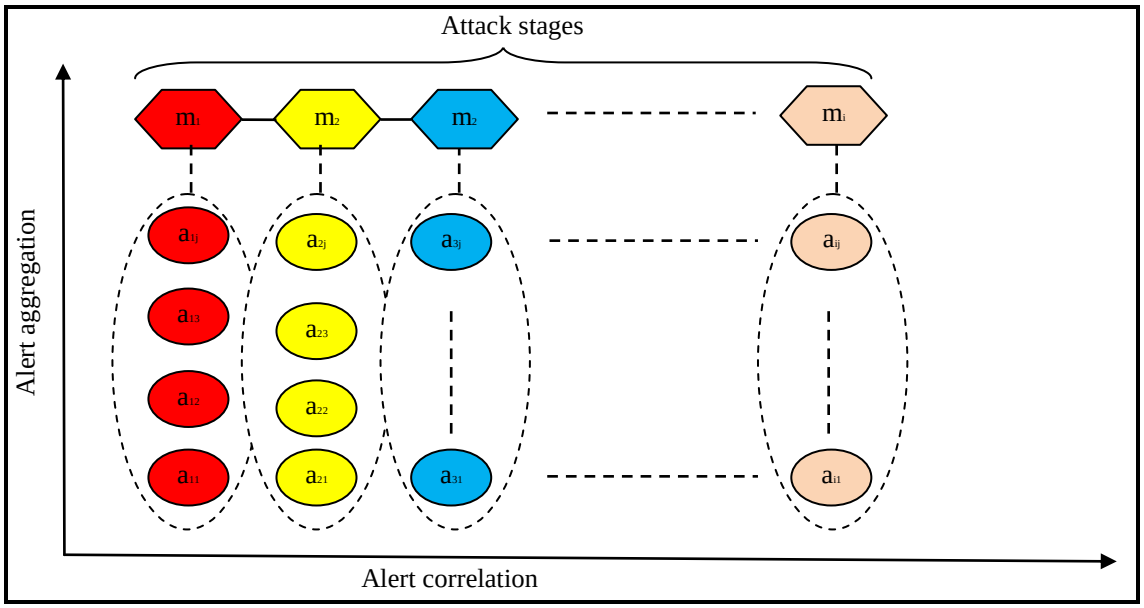

Figure 4.8 Relationships between alert correlation and aggregation.

Definition 4.4 Given a pair of attack instances a : a_1, a_2 ordered temporally in the following time slots respectively:

a_1: t_{s1} and t_{e1}

a_2: t_{s2} and t_{e2}

where t_s is the start time, and t_e is the end time.

a_1 is correlated with for a_2 if:

1- There exists at least one common capability C in $R(a_2)$, and $P(a_1)$.

2- Satisfaction of $V(a_2)$ constraints.

3- $P(a_1).t_{e1} \leq R(a_2).t_{s2}$

The proposed correlation approach consists of a series of complementary phases discussed in the following sections. A complete description of the related algorithms is given to show the system's functioning.

4.6.1 Initialization of instances of pre- and post-conditions

The objective of this procedure is to create instances of pre- and post-conditions for each alert received. Encoded conditions are in the form of corresponding capabilities based on the arguments obtained from the in-memory knowledge dictionary. Pre-condition details of previous processed alerts are deleted because they are no longer used. In other words, the remaining possible causal links of any alert are ignored as the time constraints are not satisfied.

Consider alert a_1 detected between the times t_1 and t_2, and another alert a_2 observed between t_3 and t_4, where $t_1 \leq t_2 \leq t_3 \leq t_4$. Even though a_2 has some post-conditions that match a_1 pre-conditions, they will not be correlated as a_1 is detected before a_2.

A matching between the signatures IDs in the knowledge library and those of the sequence of the received raw alert is performed. Therefore, lists of pre- and post-condition identifiers are obtained. The argument of each condition is identified and the encoded capabilities information is stored in corresponding collections in the database. Figure 4.9 shows the implemented algorithm of the creation of pre- and post-conditions details.

```
Algorithm: Pre and Post conditions initialization
Inputs: Sequence of raw alerts R, knowledge lib KLB
Output: Encoded Capability EC
Methods:
        //KLB an object of knowledge library
        // PreC :collection of Pre conditions Pre, PosC : collection of Post conditions Pos
       for i←1 to length[PreC]
           do DELETE (PreC, Pre)
       for i←1 to length[R]
        do
          if KLB.id= R.sigId
            get KLB.id, KLB.id.Pre(arg), KLB.id.Pos(arg)
        for i←1 to length[KLB]
         do
           In case of
             arg=1
                    INSERT (PreC, EC(srcIPAddress))
                    INSERT (PosC, EC(srcIPAddress))
             arg=2
                    INSERT (PreC, EC(srcPort))
                    INSERT (PosC, EC(srcPort))
             arg=3
                    INSERT (PreC, EC(destIPAddress))
                    INSERT (PosC, EC(destIPAddress))
             arg=4
                    INSERT (PreC, EC(destPort))
                    INSERT (PosC, EC(destPort))
```

Figure 4.9 Algorithm of initialization of pre- and post-conditions.

4.6.2 Knowledge initialization

A complete knowledge is initialized in memory when the MARS server starts. The total memory space of a knowledge base of 15,000 signatures does not exceed a few kilobytes. The initialization process incorporates parsing of the knowledge text file (instead of a text file, an XML representation can be used for faster processing). A

dictionary data structure is created to store knowledge details. The initialization algorithm is shown in Figure 4.10.

Algorithm: Knowledge Dictionary Initialization
Inputs: Text Knowledge *TK*
Output: Knowledge Dictionary *KD*
Methods:

```
create KD as a Dictionary
// KD.id is the Id number of each entry= Snort SigId
// KD.id.Pre is the list of preconditions associated with SigId
// KD.id.Pos is the list of postconditions associated with SigId
// create KL as array of TK lines
KL ← SPLIT TK in lines separated by (;)
for i←0 to length[KL]
    do
        In case of KL[i]
          start with ("sid") then KD.id ←KL[i].id)
          start with ("pre")
                then
                    KLI←SPLIT KL[i] separated by (,)
                    j←0
                    for j←0 to length[KLI]
                        KD.id.Pre(arg) ←KLI[j](arg)
          start with ("pos")
                then
                    KLI←SPLIT KL[i] separated by (,)
                    j←0
                    for j←0 to length[KLI]
                        KD.id.Pos(arg) ←KLI[j](arg)
```

Figure 4.10 Algorithm of knowledge initialization.

4.6.3 Correlation algorithm

The encoded capabilities stored in a collection of pre- and post-conditions are used to create the initial correlation graph, called a *temporary correlated collection*. In this

collection, all correlated elementary alerts are stored for further processing, reflecting atomic correlations. The size of the information in temporary collections may be huge, and hence graph reduction and alert aggregation functionality are performed to obtain the final graph. The correlation process is based on the satisfaction of:

- Causal relationship based on pre- and post-conditions of each detected alert.
- Temporal and spatial constrains such as IP address, port and detected time.
- Service configuration and vulnerability details.

Each correlated alert must belong to what we have called in this research *generated events*. Complete details of events are stored in a separate collection designated *InfallEventsC*. Initially, an in-memory hash table called a *correlated map* is created, and then the details are transferred to a t*emporary correlated collection*. The detected event takes the earliest start time and the latest end time among the start and end times of all corresponding alerts. An event is detected if at least two correlated alerts are detected. However, every new event is evaluated if it can be combined with other detected events on the basis of common characteristics. If there is a casual link between previous aggregated alerts and one of the detected alerts associated with the new event, the two events can be combined. In case of a connection between two events, the original event will become a master event and the new one will be considered a slave event during the process until they become a single accumulated event. The resulting event title is a concatenation of the intrusion category names of each group of events, as shown in Figure 4.11, where Attack A, B, and C are general descriptions of the attack.

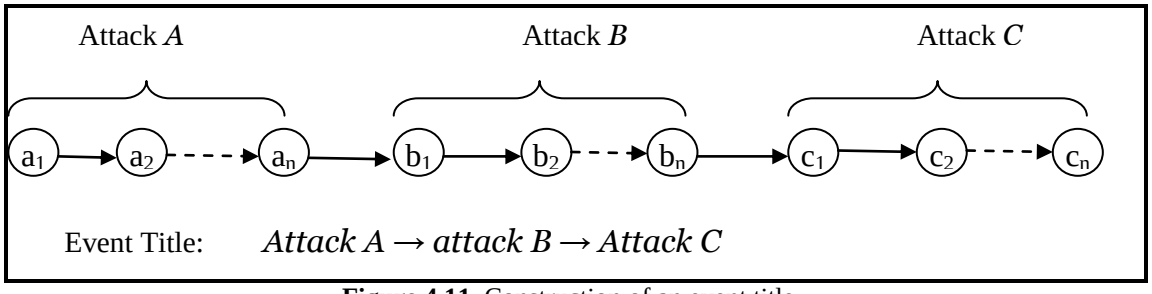

Figure.4.11 Construction of an event title.

Algorithm: Alert Correlation

Inputs: Encoded Capabilities *EC* , Vulnerability *VH*(*IP,V,P*)

Output: Correlated alerts

Methods: // Temporary Correlated Collection *TempCorrelated*

// Pre conditions Collection *PreC* , Post conditions Collection *PosC*

DELETE all rows in *TempCorrelated*

get *lastAnalysisId, lastEventId*

set *CorrelatedAlerts*←

SELECT *AlertId* **from** *PreC, PosC*

WHERE *PreC.AlertId* = *PosC.AlertId*

AND *PreC.IPaddress* = *PosC.IPaddress*

AND *PreC.Port* = *PosC.Port*

AND *PosC.endTime*≤ *PreC.startTime*

AND *VH.V is true*

If *length*[*CorrelatedAlerts*] >0

analysiId ← *analysiId+1*

/c*Map*: Correlated Map :hash table

c*Map* ←*null*

for *i*←0 **to** *length*[*correlatedAlerts*]

do

// the first alert will be the causing alert *causingA*

// the second alert will be the caused alert *causedA*

INSERT (c*Map, causingA*[*i*])

INSERT (c*Map, causedA*[*i*])

newEvent←*lastEvent*+1

for *j*←0 **to** *length*[c*Map*]

do

INSERT (*TempCorrelated,* c*Map*[*j*]. *causingA*)

INSERT (*TempCorrelated,* c*Map*[*j*]. *causedA*)

INSERT (*TempCorrelated,* c*Map*[*j*]. *newEvent*)

newEvent.startTime← **MIN**(c*Map.startTime)*

newEvent.endTime← **MAX**(c*Map.startTime)*

// InfallEvent Collection *inafallEventC*

INSERT (*inafallEventC, newEvent.fields*)

combineInfallEvent

Figure 4.12 Alert correlation algorithm.

Once all received alerts are processed and each alert is assigned to a specific intrusion event, the original alert collection is updated in order to perform alert aggregation. The algorithms of alert correlation and event generation are shown in Figures 4.12-4.13.

```
Algorithm: combineInfallEvent
SET canBeCombinedEvents←
        SELECT  infallEventId from infallEventC and aggregatedC
        WHERE infallEventC.infallEventId= aggregatedC. infallEventId
       AND  infallEvent.AlertId IN
          (SELECT  causingAlertId from TempCorrelated
            WHERE TempCorrelated.infallEventId= newInfallEventId  )
            UNION
          (SELECT   causedAlertId from TempCorrelated
             WHERE TempCorrelated.infallEventId= newInfallEventId  )
       If length[canBeCombinedEvents] >0 then
          // update the existing event
          updateCorrelated(canBeCombinedEvents[1..n], newEventId)
         else
         // create a new record in infallEventCollection
             INSERT (infallEventC, newInfallevent)
   updateInfallEvent(newInfallEventId)
```

Figure 4.13 Algorithm of two combined events

4.7 Alert aggregation

A common problem among alert correlation systems is the huge amount of atomic alerts generated by an IDS and possibly by several IDSs. An IDS may trigger a large quantity of the same alerts at close time intervals that are related to the same security violation. Alert aggregation is proposed to remove duplicated alerts, e.g. the same alerts corresponding to the same signature description or attack class. A pre-defined window is used to determine whether two alerts are close enough to be aggregated into a single alert. In addition, our aggregation approach is based on graph reduction techniques that remove duplication in vertex set and migrating connecting edges to the nominated node.

The resulting graph will only contain alerts that are in fact representing different security events.

Definition 4.5: Given a cyclic directed graph $G(V,E)$ where V is the vertex set and E is the edges set, the in-degree of a vertex is the number of edges entering it. A vertex with zero in-degree values indicates a vertex with no edges entering it (e.g. root nodes).

In the attack graph, the node's in-degree is the number of how many times the alert appears in *caused* alert group. The aggregation algorithm begins with defining the in-degree value of each node which is not aggregated in the graph. A list of zero in-degree values are identified to represent the first layer of the graph nodes; in other words, the alerts that are not caused by others. The zero in-degree list will contain groups of similar alerts occur at different times. Each group is treated as follows:

1- Nominate a master alert, which is the first alert in the temporally sorted list.

2- The aggregation process for the other alerts in the same group is based on:

- Similarity of signature IDs; this can be generalized to consider attack classes for a coarse granularity.
- Equality of source and destination IP addresses of the parties involved.
- The time difference between the detection of the two alerts does not exceed a defined value, e.g. 1 second.

3- If the above conditions are satisfied, the processed alert is added to the aggregated alerts corresponding to its master alert.

4- Change all the relationships between the aggregated alert and other alerts in the whole graph by replacing it with its master alert. Hence, the master alert will represent the aggregated alerts without losing the causal connections in the primary correlated collection.

5- Since all aggregated alerts are represented by a single alert, the corresponding time should cover the actual detection time for any further correlation. Thus, the start time of the master alert is the earliest time among the aggregated start times, and the end time is the latest one.
6- Remove the aggregated alert from the graph; however, the original information is not ignored as the graph can be disaggregated when required. Each master alert has its own counter of related aggregated alerts and graph layer.

After aggregating each group, the first graph layer, zero in-degree of all aggregated groups, is decremented by 1 to obtain the next layer. This is an opposite method to creating zero in-degree values. The second level will also have zero in-degree nodes and the same procedure is executed in an iterated fashion until all the graph nodes are treated. The algorithms shown in Figures 4.14-4.17 are describing the complete steps of the aggregation process.

```
Algorithm: AggregationAnalyzer
Input: Temporary Correlated Alerts
Output: Aggregated alerts collection
Declaration: Graph : NameValueCollection,  Indegree: hashtable,
             aggregatedAlertList:ArrayList , allRawAlerts: Hashtable
Methods:
Create a queue aggregatedZeroIndegreeAlertsCollection
Get zeroIndegreeAlertsList
Layer← 1    // the first level in the graph , initial alerts not caused by other alerts
aggregateZeroIndegreeAlertsId(zeroIndegreeAlertsList,
aggregatedZeroIndegreeAlertsCollection, Layer)
if  length [aggregatedZeroIndegreeAlertsCollection]>0
    then
     zeroIndegreeAlertsList←null
     Count← length [aggregatedZeroIndegreeAlertsCollection]
     for i←1 to Count
       do  // count denote how many groups have been aggregated
        sameAlertType← DEQUEUE  aggregatedZeroIndegreeAlertsCollection
        n←0
        while AlertId=sameAlertType[n]
           do
              if NOT (Graph[AlertId].values=null)
                 get Graph[AlertId].values
                for m←1 to length[Graph.values]
                 do
                    Indegree(values.AlertId) ←  Indegree(values.AlertId)-1
                    if Indegree(values.AlertId)=0 then
                          INSERT (zeroIndegreeAlertList, values.AlertId)
                 n←n+1
i←i+1
Layer←Layer+1
if  length[aggregateZeroIndegreeAlertsId]>0
   then
      aggregateZeroIndegreeAlertsId(zeroIndegreeAlertsList,
aggregatedZeroIndegreeAlertsCollection, Layer)
```

Figure.4.14 Aggregation analysis algorithm.

```
Function:AggregateZeroIndegreeAlertsId (zeroIndegreeAlertsList,
                    aggregatedZeroIndegreeAlertsCollection, Layer)
  if length[zeroIndegreeAlertsList]=1
    then  // in case of a single zero indegree node
      aggregatedZeroIndegreeAlerts:Array[]
      INSERT (aggregatedZeroIndegreeAlerts,zeroIndegreeAlertsList[0])
      ENQUEUE (aggregatedZeroIndegreeAlertsCollection,
                              aggregatedZeroIndgreeAlerts)
       saveAggrAlert(zeroIndegreeAlertsList[0],1,Layer)
       UpdateRawAlertCollection(zeroIndegreeAlertsList[0],
                                    zeroIndegreeAlertsList[0])
  else  // in case of zero indegree contains more than one element
      for i←0 to length[zeroIndegreeAlertsList]
        do
          if zeroIndegreeAlertsList[i]<0  // has been already aggregated
            continue
              aggregatedZeroIndegreeAlerts:Array[]
              INSERT (aggregatedZeroIndegreeAlerts, zeroIndegreeAlertsList[i])
               IndexalertId← zeroIndegreeAlertsList[i]
              appointedRawAlert: RawAlert  // nominated as a master alert
              allRawAlerts←null
         if allRawAlerts contains IndexalertId
           then
             appointedRawAlert=allRawAlert[IndexalertId]
           else  // create a new one
              appointedRawAlert← new RawAlert[IndexalertId]
              ADD (allRawAlerts, appointedRawAlert)
              UpdateRawAlertCollection(appointedRawAlert,
               appointedRawAlert)
rawCollectionCount: integer
        // internal loop
          for j←i+1 to length[zeroIndegreeAlertsList]
            if zeroIndegreeAlertsList[j]<0
              continue
```

Figure.4.15 Aggregation of zero in-degree alerts algorithm.

```
                    // new alert in the same aggregated group
                     indexRawAlert:RawAlert
                     internalAlertId :int
                      if allRawAlerts contains  zeroIndegreeAlertsList[j]
                        then
                            indexRawAlert←allRawAlerts[internalAlertId]
                         else
                            indexRawAlert←new RawAlert[zeroIndegreeAlertsList[j])
                            ADD (allRawAlerts, indexRawAlert)
                           if appointedRawAlert.sigId =indexRawAlert.sigId
                             AND appointedRawAlert.srcIPAddress=
                                                 indexRawAlert.srcIPAddress
                            AND appointedRawAlert.desIPAddress =
                                                 indexRawAlert. desIPAddress
                       AND DIFF (appointedRawAlert.endTime,
                                          indexRawAlert.endTime)≤1
                 then
                      INSERT (aggregatedZeroIndegreeAlerts, zeroIndegreeAlertsList[j])
// relationships will be shifted to the new master alert
              if NOT(Graph.zeroIndegreeAlertsList[j])=null
                 then
                      values← get Grpah.zeroIndegreeAlertsList[j]
             for k←0 to length[values]
                 do
                      INSERT (Graph.values, indexAlertId)
                     // remove the aggregated alert from the Grpah
                      DELETE (Graph.values , zeroIndegreeAlertsList[k])
                    // add the aggregated alert to the aggregated list
                     INSERT (aggregatedAlertIdList, zeroIndegreeAlertsList[k])
                   // make this Id as a negative value in the zero indegree
                    zeroIndegreeAlertsList[k] ← zeroIndegreeAlertsList[k] × (-1)
                   // increase the alert count for the associated master alert
                   rawCollectionCount ← rawCollectionCount+1
```

Figure.4.16 Aggregation of zero in-degree alerts algorithm (continued).

```
// the aggregated group of alerts takes the start time of the earliest alert
// and the end time at the latest end time
if appointedRawAlert.startTime > indexRawAlert.startTime
   then
      appointedRawAlert.startTime← indexRawAlert.startTime
if appointedRawAlert.endTime < indexRawAlert.endTime
   then
      appointedRawAlert.endTime← indexRawAlert.endTime
// delete the post conditions of the aggregated alert
DELETE (postConditionTable, indexRawAlert)
ENQUEUE (aggregatedZeroIndegreeAlertsCollection,
                aggregatedZeroIndgreeAlerts)
```

Figure.4.17 Aggregation of zero in-degree alerts algorithm (continued).

4.8 Graph reduction

In order to reduce the complexity of the resulting graph, data redundancy should be eliminated. The graph consists of nodes representing aggregated alerts and edges representing the casual relationships. The number of nodes is not affected while the number of edges is minimised without affecting reachability. Hence, the target is to find a minimal DAG with the least number of arcs and which is equivalent to the original DAG. Consider the case shown in Figure 4.18, with four alerts: *a, b, c,* and *d*. If Alert *a* is causing Alert *b* and *b* is causing *c*, there is no need for the transitive edge between *a* and *c*, and similarly the edges between *a*-*d* and *b*-*d*. The removal of the transitive optional edges does not have any effect on connectivity between the original nodes.

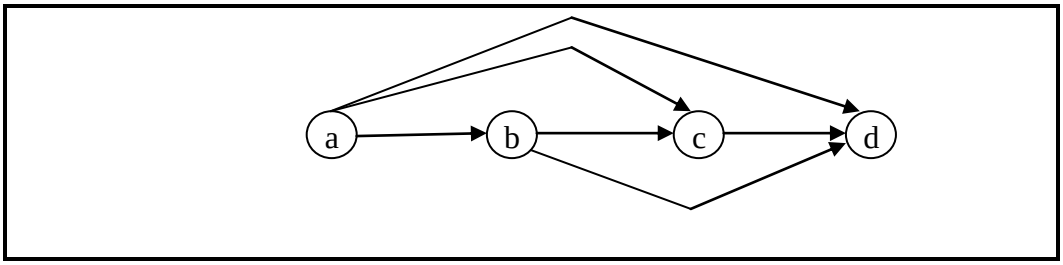

Figure 4.18 Transitive edges in graph.

This is based on the assumption that the relationships between nodes can propagate and the removed edges are considered optional.

Definition 4.6: given a *DAG G=(V,E)*, *V=X* is the vertex set, *E=R* is the set of arcs of the graph, let *n=#V*, *V*={1,….,n}, the reduced graph *G′(V,E′)* is a *DAG* with the following properties:

(1) The vertex set (*#V*) of *G(V,E)* is equal to the vertex set (*#V*) of *G′(V,E′)*.

(2) The directed paths between the vertex in *G(V,E)* and *G′(V,E′)* are similar.

(3) *G′(V,E′)* has the smallest number of edges *E′=R′* between vertex sets without affecting the connectivity, *R′<=R*.

Two algorithms have been developed: online graph reduction for edge deletion on the left side of the graph, and offline graph reduction for edge deletion on the right side of the graph. The online algorithm removes the transitive edges at the real-time when every node joins the graph. This procedure is performed at the first stage of correlation and before alert aggregation in order to minimise the system's processing time. The offline algorithm results in a further graph reduction if any redundant connection exists after the graph is built, starting from the leaf nodes to the root nodes. To clarify the idea, consider the alerts correlated by the system in the initial stage shown in Figure 4.19 below:

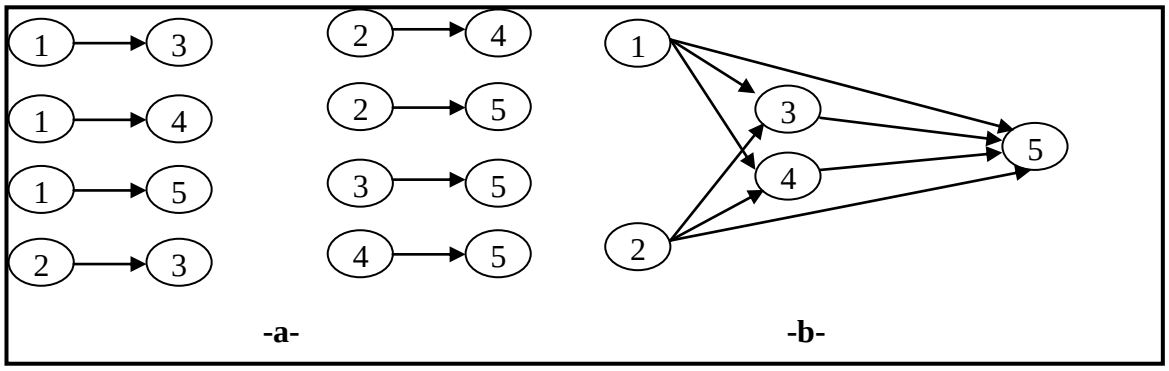

Figure 4.19 Example of graph reduction.

There are five nodes and eight edges connecting these nodes to represent the causal relationship. In Figure 4.19 (a), the number of the representing nodes n is half the number connecting arcs $\#V$. The edges $1 \rightarrow 5$ and $2 \rightarrow 5$ can be deleted because they are redundant and the description of the intrusion sequence will not be affected. In the proposed reduction algorithm, each node has two lists of children and parents, and the aim is to remove the duplicates in these lists as shown in the following two algorithms displayed in Figures 4.20-4.23

Algorithm: OnlineGraphReducer

Input: Correlated Graph

Output: Reduced Correlated Graph

Declaration: GraphNode: <id, value>

Parents , Childs,Roots: List of GraphNodes

Ancestors: Dictionary of GrpahNodes<GraphNode,List of GraphNode>

NodeSet,: Dictionary of GrpahNodes<int, GraphNode)

node1**,** node2 : GrpahNode

Methods:

// perform for each edge, if the n is the nodes number, the edge will be n/2

for *i*←0 **to** *length*[*nodes*]/2

do

// the node on the left side, causing alert

node1id←*nodes*[*i*]

//the node on the right, caused alert

node2id←*nodes*[*i*+1]

node1←*null*

//nodeSet is the resulting set after reduction

if *nodeSet* contains *node1id*

then

// if it is already added to the nodeSet

node1←*nodeSet*[*node1id*]

else

//otherwise create a new GraphNode and ancestors list for node1

node1← **new** *GraphNode*(*node1id*) ;

a*ncestors*[*node1*] ←**new** *List of GraphNode*

//node2 is processed similarly

Node2←*null*

if *nodeSet* contains *node2id*

then

// if it is already added to the nodeSet

Node2←*nodeSet*[*node2id*]

else

//otherwise create a new GraphNode and ancestors list for node2

Figure 4.20 Online reduction algorithm.

```
node2← new GraphNode(node2id) ;
        ancestors[node2] ←new List of GraphNode
//check all parents of node2, if one exists in node1's parents, remove it from
node2's parents to avoid duplicates (transitive relationships)
for k←0 to length[node2.parents]
    do
      if ancestors[node1] contains node2.parents[j]
       then
          DELETE (node2, node2.parents[j])
//add node2 as a child of node1
INSERT (node1.child,node2)
//add node1 to node2's ancestors if it is not already existent
if  NOT (ancestors[node2] contains node1)
  then
     INSERT (ancestors[node2], node1);
// add all ancestors of node1 to ancestors of node2
for j←0 to length[node1.ancestors]
    do
      n: GraphNode
      n←node1.ancestors
      if NOT (node2.ancestors contains n)
      INSERT (node2.ancestors,n)
 // if node2 is a root node remove it from roots because it is not root anymore
//after being  a child of another node
if node2 ∈ roots then DELETE (roots, node2)
// if node1 is not already in roots add it to roots
if  length[node1.parents]=0 AND NOT (node1∈ roots)
  then
    INSERT (roots, node1)
```

Figure 4.21 Online reduction algorithm (continued).

```
Algorithm: OfflineGraphReducer
Input: Correlated Graph
Output: Reduced Correlated Graph
Declaration:
 GraphNode: <id, value>
 Parents , Childs, Roots, n, grandson: List of GraphNodes
 indirectedOffSprings : Dictionary of GrpahNodes<GraphNode,List of  GraphNode>
Methods:
for i←0 to length[roots]
     do
        // if the root node is already existent in sons group return the group
        if n ∈ indirectedOffSprings
          then  return indirectedOffSprings.n
// if the root node does not have any child create a new list
if  length[n.child] =0
  then
       return new list of GraphNode
 for i←0 to length[n.child]
     do
       //check the sons of the sons of each node in roots
        for j←0 to length[n.child[i].child]
          do
              if grandson∈  n.child[i].child
                then
                     // if this son is not a member of the sons group add it
                    if NOT grandson∈  indirectedOffSprings
                    INSERT (currentIndirectedOffSprings, grandson)
         indirectedOffSprings.n=currentIndirectedOffSprings
for k←0 to length[indirectedOffSprings]
     do
       if n ∈ indirectedOffSprings
         then
              // remove duplicates in sons of the nodes on the same sequence
```

Figure 4.22 Offline reduction algorithm.

```
l←o
  while l<length[n.child]
      do
        if n.child[l] ∈ indirectedOffSprings.n
            then
                DELETE (n.child,n.child[l])
             else
                l← l +1
```

Figure 4.23 Offline reduction algorithm (continued).

4.9 Prediction of undetected intrusion

Beyond the correlation function's primary role of reducing information complexity, it may also handle unobserved attack activities and estimate the attacker's planned path to achieve its goal. Intrusion activity is considered a planning activity, because the planning actions are explained by their pre-conditions and effects. In this sense, correlation functionality is used in discovering unobserved alerts (false negatives) and predicting intrusion intention. Missed alerts have been one of the major limitations of IDSs, particularly signature-based ones, and this is caused by three reasons:

-The intrusion action is unknown (e.g. 0-day attack), and the IDS has no knowledge of the attack.

-The attacker performs certain evasion techniques to deceive signature-based IDSs, which may only be other variations of known existing attacks.

-In high-speed network environments, as is the case in current systems, the IDS is unable to keep up with the received traffic. As has been observed in Chapter 3, IDSs drop traffic packets when they are overloaded.

In an alert correlation context, missed alerts can result in broken attack scenarios, dividing the attack graph into different sub-graphs. Furthermore, unobserved attacking

sequences can lead to an incomplete graph if the IDS misses some alerts comprising late attack stages. Hence, the objective of the attacker will be uncertain for the observer. In such cases, alert correlation generally uses abduction techniques [49] to estimate the missing data and the intruder's intention in partial attack sequences.

4.9.1 Alerts missed by IDSs

Several efforts have been presented to overcome the limitation of alerts missed by IDSs. Most of them focus on how to repair the broken scenario by constructing all possible actions using attack libraries [35, 131]. Consider *Attack 1* and *Attack 2* in Figure 4.24, where *Attack 1* consists of three alerts (a_1, a_2 and a_3) and *Attack 2* involves two alerts (a_5 and a_6). The two attacks actually belong to the same intrusion sequence. The IDS misses alert a_4, causing two separate attack scenarios. In order to reason about the missing middle alert and connect the two attacks, certain virtual nodes are created along the attack path between a_3 and a_5. In our correlation approach, the causal link between the nodes is based on the equality constrains of the capabilities in the pre-conditions of node a_5 and any possible node with similar capabilities in its post-conditions. Similarly, a match between capabilities defined in the post-conditions of a_4 and capabilities defined in the pre-conditions of any possible nodes, is determined to link these nodes to a_3.

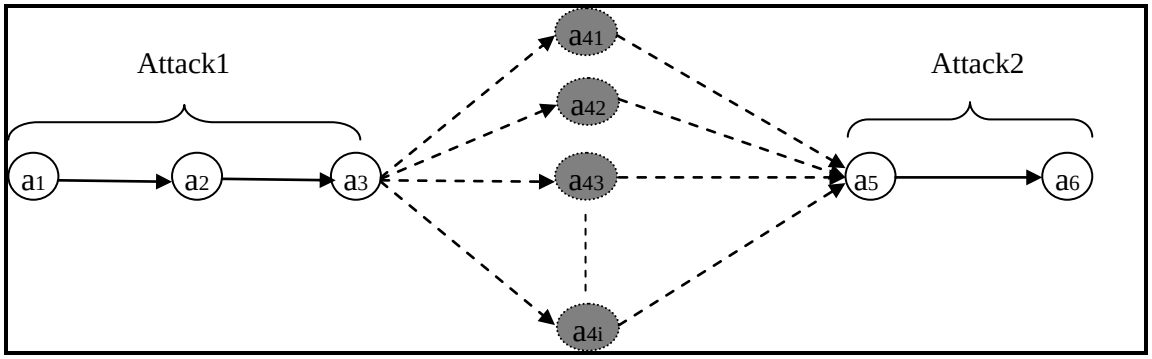

Figure.4.24 Reasoning about missed alerts.

Definition 4.7: a sequence of actions $a_1,a_2,.....,a_n$ comprising an attack plan and two sub-sequences of observed actions $a_1,a_2,.....,a_k$ and $a_l, a_{l+1},....., a_n$ are linked virtually as candidates of the same attack scenario if:

a. There is at least one action node (virtual node) sharing at least one capability in the post-conditions of a_k and pre-conditions of a_l
b. Satisfaction of temporal, spatial and vulnerability constrains.

The missed alerts can be more than a single action node, so all possible matched nodes are searched forwards starting from a_k and backwards starting from a_l. The algorithm progresses until a match is identified to link the two attacks. The virtual nodes can be a series of nodes in a sequence starting from the last action node in the first half of the broken attack scenario to the first action node of the consequent other half. However, the number of estimated nodes can be large and this will add more complexity to the resulting graph.

In fact, the repair of broken-scenario approaches may add more complications. A considerable amount of processing power is consumed and this is critical for online applications. The idea of attack generalization presented in section 4.4 can give the administrator a general view of the actual attack without having to rely on identifying the exact missed alerts. We have adopted the idea of reasoning about missed alerts by generally giving the attack category instead of a potentially infinite number of virtual alerts. Some of the reasons behind this are:

1- As stated earlier in this section, alerts can be missed as a result of one of two reasons: unknown attacks or missed attacks due to performance issues. The second category can be estimated using virtual nodes and edges based on a knowledge library. However, if the attack is a 0-day or a new variation of a known attack, the specific reasoning about unobserved alerts will not give the

details of the exact intended attack actions. Hence, virtual nodes do not represent the actual alerts but similar alerts, which can mislead the administrator by producing false positives.

2- Based on the initial assumptions of this thesis, coordinated attacks consist of a number of steps that are not isolated. The absence of some of these irrelevant steps, which are usually much less compared to the related steps observed, does not affect the correlation approach. Once again, the generalized formalisation of capabilities can assist in building the attack graph even with the use of incomplete attack knowledge. For instance, consider the attack stages associated with the scenario shown in Figure 4.25. The link shown in red denotes a generalized capability in the capabilities set. To be more specific, assuming the attack steps are for SQL injection (SQLI) stages, all alerts involved in this attack share the *SQL injection* capability, coloured in red. The other capabilities coded in different colours are specific to certain other conditions. Hence, even if alert a_3 is missing or not covered by the operating IDS, the causal link is still established using the general specified condition.

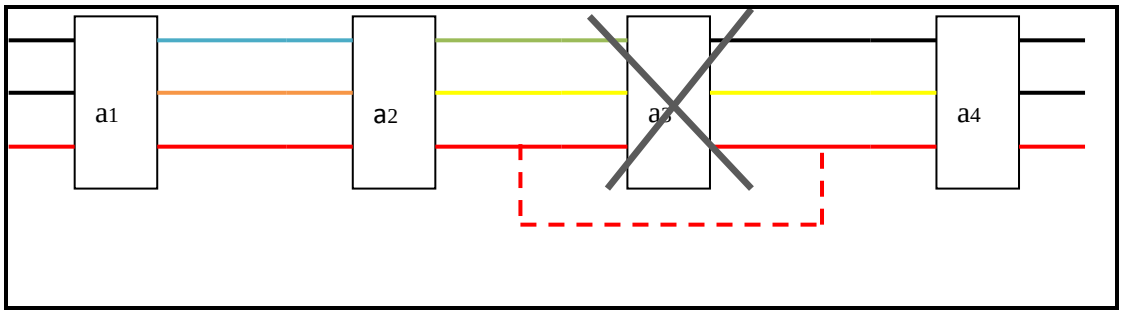

Figure 4.25 Reasoning about missed alerts by generalised capability formalisation.

3- The main aim of the alert correlation and aggregation function is to build an attack graph with minimal data and to reduce the false positive rate. Therefore, the process of generating further information, which could result in false positives, conflicts with the main concept of alert correlation. Therefore, our

reasoning revolves around the attack scenario rather than specific potential alerts. In other words, the focus is on discovering the intention of the attack and recognizing the intrusion plan.

4.9.2 Intruder intention recognition

Intruder intention recognition is the task of inferring intrusion goals from the observation of intruder actions or the consequences of these actions [49]. Intrusion actions are described in terms of conditions required to achieve actions and the conditions provided as a result. Observed actions are ordered temporally to constitute an intrusion plan. Hence, intention recognition is a prediction task to identify intrusion goals from a partial set of observed actions. Taking the case of certain alerts constituting an intrusion plan detected by an IDS, the aim of the system would be to predict future incoming alerts along the intrusion path.

Plan recognition in an intrusion context is different, because attackers try to hide their activities and identities [163]. The general form of plan recognition assumes that the actor follows a series of complete and ordered sets of actions to achieve a specific goal, which is not the case in intrusion behaviour [164]. Adversarial recognition involves dynamic actions and goal changes based on identified effects. For example, an attacker intending to break into a system for a specific vulnerability may change to another intention if a new vulnerability is discovered that may provide more control over the attacked system. However, in general the ultimate goal of the attacker can be predicted – i.e. full access to the target system, even if the behaviour is dynamic.

Typically, the prediction process is based on the same notion discussed in the previous section. Estimated virtual causal edges and virtual nodes are created based on the pre- and post-conditions of the observed actions. However, a large number of attack paths can be recognised, building a huge attack graph. The virtual node creation is limited to

the first malicious activity detected if some of the related actions have already been detected. For instance, scanning behaviour is considered suspicious activity, so it cannot be predicted what will come next as there is a large number of paths. However, from the target system knowledge the attack paths can be bounded to a lesser number. Therefore the algorithm progresses to construct virtual paths until a malicious activity is recognised. The determination of the malicious level of an activity is based on the priority information provided by the rules.

However, as mentioned in the previous section, our approach of defining a layered structure of capabilities and an attack classification assists in recognizing intrusion intention. Identified capabilities associated with malicious actions are used directly to express the intrusion goal, while suspicious actions can contribute to the achievement of the intrusion goal. For example, an alert has three post-conditions represented in the form of capabilities as follows: *stored procedure*, *PHP injection* and *SQL injection*. The most generic capability, *SQL injection,* is used to recognise the intrusion goal and is classified as an SQLI attack plan. In this context, the administrator can take a reactive response to prevent this attack before it can achieve its final goal, such as modification of the database in the target server. Recognition of attack intention is considered as identifying that a group of actions is a subset of another group. It is assumed that the attacker has completed all attack stages, and that the task is to find any subset of these stages that represents a potential attack.

Definition 4.8 : let a is an attack consisting of a series of candidate steps $a_1,a_2,.....,a_n$, and a' is an attack consisting a subset $a_1, a_2, ...,a_m$. a', which is to say a subset of a if at least one action from a' shares the same generic capability of at least one action from a and satisfies the temporal, spatial and vulnerability constraints.

Example: suppose attack *a* consists of five aggregated alerts, with the first two alerts belonging to scanning behaviour and the remaining alerts relating to an SQLI attempt. Another attack, a', consists of three alerts, the first two being a sign of scanning and the third classified as an SQLI attack (SQLIA). Both attacks share the same temporal and spatial attributes. It can be predicted that a' is a subset of a, as they share at least one generic capability. However, the disadvantage of this approach is that two different attackers not cooperating will be considered one and the same.

4.10 Conclusion

The main objective of alert correlation systems is the identification of the multi-stage attack which may be discovered from analysis of the IDS alerts. These alerts have certain features that can be used to detect causal relationships between temporally distributed activities. In this chapter we have presented the core concept of our reasoning framework for alert correlation to address the problem of detection of coordinated attacks. MARS framework has been detailed involving multiple cooperative components.

We have defined the underlying principles of our framework based on *provides/requires* model. A combined analysis of IDS's alerts and description of attack classes are used to derive the pre- and post- conditions of each received alerts. A scheme to represent our knowledge base has been described using a hierarchal and a multilayer classification. Vulnerability modelling is used to support alert verification in order to reduce the generated attack graph. The generalisation concept is utilised to predict attack intention. A detailed description of the algorithms involved has been presented as well as the relationships between the system components. Aggregation and graph reduction approaches are also used to obtain the resulting attack events in a manageable graph.

CHAPTER 5: MARS FRAMEWORK IMPLEMENTATION

5.1 Introduction

The previous chapter has presented the fundamental basis of the proposed framework for the multi-stage attack recognition system. In this chapter, the MARS framework is practically implemented to evaluate the proposed algorithms discussed in Chapter 4. In this chapter, the general architecture of the system is described. Multi-stage attack recognition as an alert correlation functionality is a multi-task process. Each task is performed by a corresponding component in a sequential manner. Moreover, the design details are presented to illustrate how the algorithms are elaborated to obtain the results.

5.2 MARS components

The objective of the proposed system is to construct an overview of the security status of the system under attack. This functionality consists of a sequence of components, with each component responsible for a task and the result of each task supplied to the next component. Figure 4.1 in the previous chapter presented the design components and the communication between these components. Figure 5.1 shows the workflow of the system process starting with the receipt of alerts from the IDS sensors and ending with the administrative console.

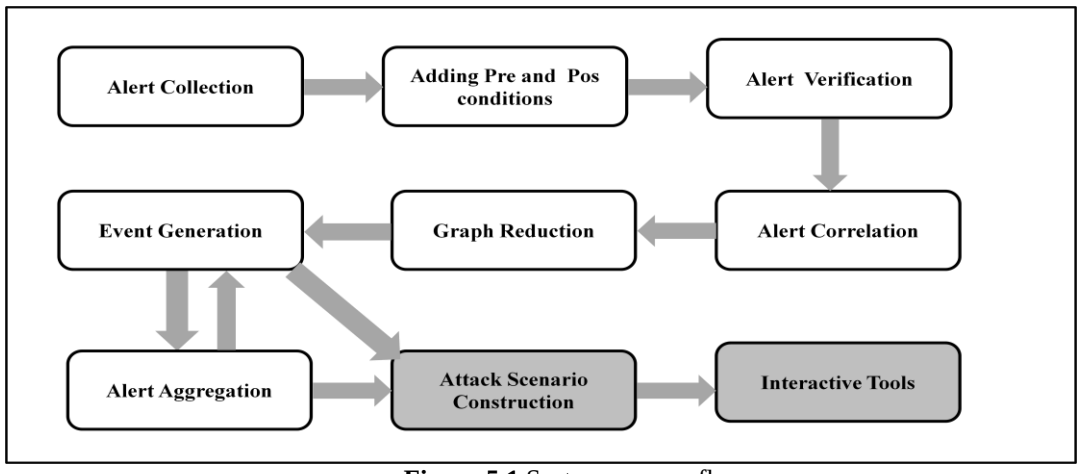

Figure 5.1 System process flow.

5.2.1 Alert collection

Since the main input of the system is the alert stream generated by an IDS sensor, these alerts need to be translated to a generic format. The alert collection component can receive alerts from different sensors in various formats. The typical format used in this respect is the Intrusion Detection Message Exchange Format (IDMEF) [116], which is considered the industry standard. In this thesis, multiple sensors to feed the system with alerts are not considered due to space limitations and because the implementation of such a system is straightforward. Alerts are converted to the standard format and stored in the MARS database for effective information search. The standardization is performed by an IDS interface (e.g. Snort interface). If a different IDS sensor is used, a corresponding interface is required, but the details of this are beyond the scope of this thesis.

In our framework, sensor-specific information is converted into attributes and values usable by the framework components. The alert names are taken from Snort database which are based on vulnerability standards such as CVE and Bugtraq. Each raw alert is translated into a standardized alert format and copied to the appropriate fields. The attributes contained in the resulting format are shown in Table 5.1

Table 5.1 Description of alert attributes.

Alert attribute	**Description**
Alert ID	A unique ID identifying the alert
Sensor ID	A unique ID identifying the IDS sensor
Start Time	The time when the attack occurs
End Time	The time when the attack ends
Source IP Address	The source IP address of the detected activity
Source Port	The source port of the detected activity
Destination IP Address	The destination IP address of the detected activity
Destination Port	The destination port of the detected activity
Signature ID	A unique ID identifying the IDS signature
Signature Priority	The severity level of the detected signature

Protocol Type	The protocol name used in the attacking activity
Master ID	The representative alert ID that represents some aggregated alerts
Infall Event ID	The ID of the corresponding generated event

5.2.2 Adding pre- and post-conditions

The core of the alert correlation mechanism is the mapping of an elementary alert to its pre- and post-conditions. This is to discover any possible relationships between the alerts in order to identify the attack patterns. Instances of alerts are created in the database with their attributes (IP address, port number and timestamp). All alerts from the previous component are processed, as they all are candidates for involvement in the alert correlation task.

The collection of encoded capabilities is constructed using the capability knowledge by assigning each alert to its pre and post conditions. For example, the alert 123 shown in Figure 5.2 as specified in the knowledge base, has two pre conditions i.e. 100 and 101, and two post conditions i.e. 200 and 201.

Sid:123; pre:100(3);pre:101(3);pos:200(3);pos:201(3)

Figure 5.2 An example of the capability knowledge base specification.

Two lists are created for each condition set with the format:

Capability ID (IP Address) , for example: **100(192.0.0.1)**

The initialization of capabilities sets (pre and post conditions) is implemented to make the linking between related alerts faster.

5.2.3 Alert verification

IDSs cannot determine whether the occurring attack is likely to be successful. Failed attacks in typical cases do not provide further information, because the attacker will find another vulnerability to exploit. Running services and vulnerability details gathered by

scanners such as Nessus [128] are used to filter alerts. Hence, the amount of processed alerts will be reduced to achieve more accuracy. The administrator feedback concerning certain attacks can be considered in this respect to achieve more accuracy. Furthermore, the false positive rate is the main concern of such systems, so alert verification contributes to achieving lower rates of this measure.

When the correlation process receives false positives as input, the quality of the results can be degraded significantly. The goal of the alert verification component is to remove alerts that do not represent true attacks. Hence the correlation rules are extended with the success of the occurring attack. In the implementation of MARS, a passive verification technique is used to provide a higher performance. This is based on the assumption of being network and hosts states don't change frequently over short period of time. Nessus scanner is used to scan the protected network to collect all required information such as: network configuration, host configuration, running services, and detected vulnerabilities. These data is stored in the vulnerability knowledge base to be applied in cooperation with the correlation algorithm.

5.2.4 Alert correlation

This is the component that implements the proposed correlation algorithm based on the pre- and post-conditions of alert instances. The correlation function is performed only for alerts that satisfy the conditions. Isolated alerts which are not logically connected are saved in the primary correlation container for any further observation.

Alerts with equivalent attributes and occurring in a certain temporal proximity are linked if they satisfy the matching between encoded capabilities. Alert ***a*** can be considered as causing alert for alert ***b*** if ***a*** occurs before ***b*** and a matching between ***a*** post conditions and ***b*** pre conditions. The matching criteria are more relaxed to

maximize detection coverage and that using the generalisation concept in attack modelling. Furthermore, there is no restricted time window to correlate pair of alerts in contrast to other proposed systems that are defeated easily by slow-and-low attack. However, the administrator has the facility to close any detected event when he/she makes sure that there is no further related activities e.g. running service is stopped. The correlated alerts which are actually the correlated master alerts (representative of aggregated alerts) are saved in the correlation collection.

5.2.5 Graph reduction

Generally, the process of graph complexity reduction involves removing some redundant graph nodes and edges while keeping the structure of the sequences of the attacks. The objective of this component is to remove the transitive edges that represent duplicates in the correlation process. The hierarchical approach, which involves generalization in relationship discovery, can cause additional links between alerts.

To keep alert processing to minimum, removing redundant edges is performed during the initial sage of the correlation process. This is based on the concept that the relationship propagate form parent to children nodes. However, the removal of edges does not affect any loss of data, as the logical connections are specified in layers.

5.2.6 Event generation

This is the main component that describes the multi-stage intrusion details. Based on the output of alert correlation, a new entry is created for each detected event. However, its data is updatable based on the aggregation results of both alerts and events. Any upcoming event is compared with previous ones to check for any merging opportunity. New events are generated as independent event if at least two alerts are correlated. Then, the system checks for any attribute matching with the previous event in addition

to any detected logical link based on capabilities information. If a matching is detected the two events are combined as single event and the previous event is identified as a master event. The event remains open until it is closed either by the administrator or a defined time window. The interactive administrative tools provided by the system allow the administrator to update or close any open events. Table 5.2 shows the information included in each event.

Table 5.2 Events information.

Field	Description
Event ID	A unique event identifier
Start time	The earliest start time among involved alerts
End time	The latest end time among involved alerts
Title	The event title constructed by the names of the involved attacks
Alert count	Number of involved alerts
Priority	Importance level of the event based on the severity of involved alerts
Closed	To identify if the event has been identified and closed
Master ID	The ID of the master event if this event has been combined with other event

5.2.7 Alert aggregation

The alert aggregator component maintains the resulting correlated alerts to minimise the redundancy. If a group of alerts share the same source and attack class, it is practical to keep only a representative alert and remove any duplicates. This task is done after correlation so as not to overload the system by aggregating isolated alerts. The main task of this component is to remove redundancy in graph nodes which are representing the same attack. There are two levels of aggregation: one is based on the attack signature and the other is to extend the aggregation to include all alerts classified as the same attack types.

The aggregation task is performed for all correlated alerts by assigning the first detected alert as a master alert. The links between the other alerts (the aggregated ones) are

replaced links to the master alert. The start and the end times of the master alert are changed to become the earliest and latest times among the aggregated alerts. However, no data will be lost as all the information is saved in the database and the disaggregation function is available.

5.2.8 Attack scenario construction

The purpose of the attack scenario construction component is to identify high level attack patterns that are composed of several individual attacks. For example, consider an intruder who first scans a victim host, then breaks into a user account on that host, and finally escalates privileges to become the root user. The three steps should be identified as belonging to one attack scenario. The attack scenarios are generated as graphs composed of nodes and edges: nodes represent the attack name and edges represent the logical relationships. The scenario graph is displayed as a summary of the attacker activities after performing aggregation and graph reduction to remove redundancy. Hence, the final goal of the developed system is to recognise attack stages connected in a temporal sequence. The results of event generation and aggregation are expressed in an attack graph to describe the attack scenario. An overview of the attack situation is displayed for the administrator in the form of an interactive graph. The detailed information of each entity in the attack graph can be navigated by the provided administrative tools.

5.2.9 Interactive tools

The interactive administrative tools are used to provide a dynamic platform. The attack scenario is presented as a graph of nodes. To save space, the nodes only show the alert name, and other details can be retrieved using the available tools. Reports and results of statistical analyses can be also accessed using these tools. Examples of some functions can be executed through this component:

- Close the open events and that after the administrator makes sure that these events will not be used any longer to be combined with other events. This mechanism is more reliable than using time windows in addition that the multistage attack activities are not frequent to overwhelm the administrator with huge amount of administrative tasks.
- Support the knowledge base with the administrator experience. For example, some new unknown attacks which are not identified in the knowledge base can be added as temporary or permanent rules.
- Provide the facility to combine similar events which are not aggregated by the system. For example, some attackers try different attack attempts looking for some holes in the victim machine.
- Block IP addresses and send notification about suspicious activities.

5.3 MARS architecture

This system is composed of four major sub-systems: a MARS server, a MARS client, sensor interfaces (e.g. Snort or Nessus) and a MARS database, as shown in Figure 5.3.

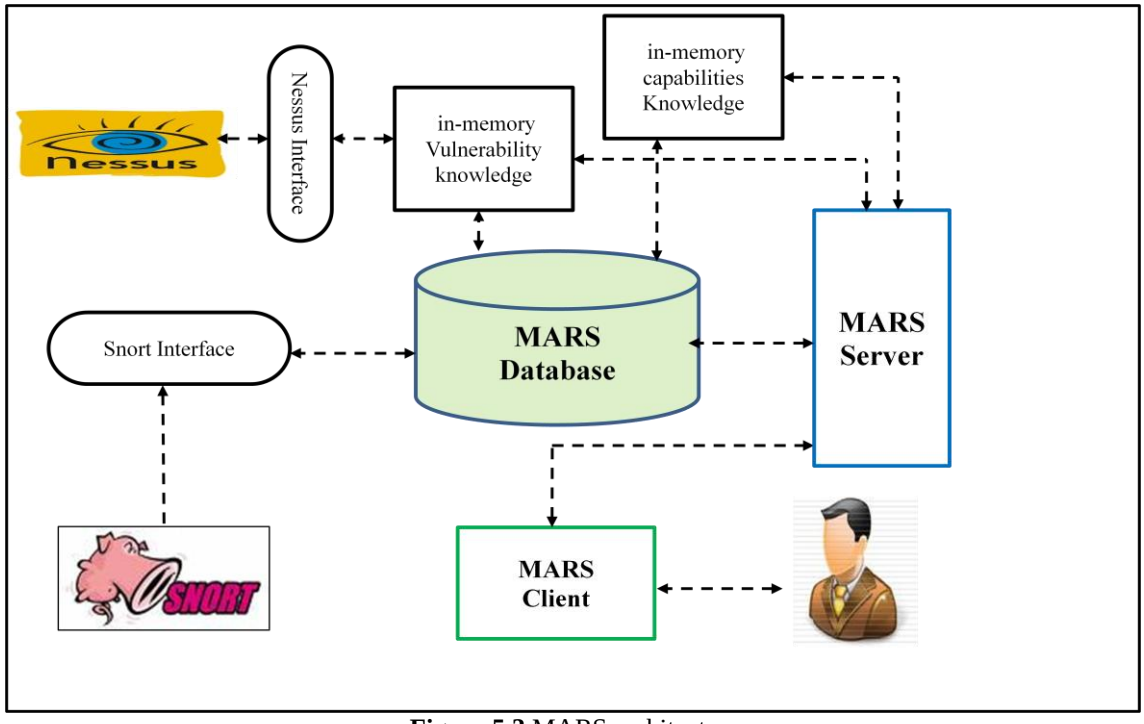

Figure 5.3 MARS architecture.

These sub-systems can be installed on different machines to take advantage of distribution. It has been observed in Chapter 3 the limitations of the performance of IDSs, as they require plenty of computational resources. Hence, the MARS system has been designed as a stand-alone system running separately from the IDS to avoid any impact on the IDS performance. In addition, the database storage to record all alerts provides the system with the ability to function as a forensic tool. The server is running as a daemon or a service in the background.

The diagram also shows the communications between the sub-systems. When the system starts, two in-memory knowledge bases are created: the capabilities knowledge obtained from the attack-defined knowledge, and the vulnerability knowledge supplied by the vulnerability scanners, such as Nessus. These details are kept in-memory to allow for faster communication and also because they are used frequently. The whole system is implemented as an object-oriented design to provide modularity and dynamic data-structure instantiation. We have made use of C++ programming language to manipulate its efficiency.

The MARS database is the core system storage where the received alerts are handled and stored. The correlation, aggregation and event generation results are also saved in the database. The interaction with the database is kept to the minimum, as we have manipulated the data structures provided by C++ for memory execution. Indexing is used for faster access and SQL queries are hardcoded using C++ commands to improve execution. The structure of the main database tables and related fields is shown in Figure 5.4.

RawCollection
AlertID
SensorID
StartTime
EndTime
SrcIPAddress
SrcPort
SrcIPSID
DestIPAddress
DestPort
DestIPSID
SigID
SigPriority
ProtocolType
StatusID
MasterID
InfallEventID

Rules
RuleBasedID
SigID
SigName
SigProtocol
SigPriority
SigCategoryTypeID
SigDirection

PostCon Set
AlertID
EncodedCapability
StartTime
EndTime
Arguments
StatusID
IPSID
Port
IPAddress
CapabilityID

PreConSet
AlertID
EncodedCapability
StartTime
EndTime
Arguments
StatusID
IPSID
Port
IPAddress
CapabilityID

CorrelatedAlertsTemp
CorrelatedAlertID
CausingAlertID
CausedAlertID
AnalysisID
VirtualEdge
InfallEventID
SlaveInfallEventID

InfallEvent
InfallEvent ID
StartTime
EndTime
Title
Treated
TreatedTime
AlertCount
Priority
Description
Closed
ClosedTime
Supervisor
AnalysisID
MasterID
PostAnalysisID
HistoryCount
CorrelatedCount
TypeID

AggrRawCollection
AlertID
SensorID
StartTime
EndTime
SrcIPAddress
SrcPort
SrcIPSID
DestIPAddress
DestPort
DestIPSID
SigID
SigPriority
StatusID
RawCollectionCount
InfallEventID
PostAnalysisID
Layer
NodeCategeory

CorrelatedAlerts
CorrelatedAlertID
CausingAlertID
CausedAlertID
AnalysisID
VirtualEdge
InfallEventID
SlaveInfallEventID

AggrCorrelatedAlerts
CorrelatedAlertID
CausingAlertID
CausedAlertID
AnalysisID
VirtualEdge
InfallEventID

Figure.5.4 Main database tables.

- *RulesCollection* is a container of all available Snort VRT signatures, bleeding edge signatures, community signatures and our developed signatures. Signature ID and signature name is a mapping of the information in the original Snort signatures. Each signature has a priority field for assigning the degree of severity of the detected attack.

The main purpose of the rules collection is the classification of each received alert based on the techniques described in Chapter 4. The intrusion category details play a main role in specifying the attack scenario and in predicting the possible undetected behaviour. Signature direction denotes attack direction, where 0: source address, 1: destination address, and 2: bidirectional.

- *InfallEventCollection* contains the history of all detected events. An initial record is created once a new event is observed. The title of the event is constructed from the sequence of intrusion categories. The event remains open for any further joining alert until it is closed based on either administrative action or a configurable time period. To detect slow-and-low attacks, this period of time can be maximised without affecting the system's performance.

- The final correlation results are stored in the *CorrelatedAlerts* container, which holds a record for each correlated and aggregated events. If the alert belongs to combined events, the details of the master and slave events are recorded. Any further aggregation will be presented in the *AggrCorrelatedAlerts* container.

MARS Client is a sub-system that provides tools offering interactive administrative tasks. The graphical user interface is implemented here using the aid of commercial tools, namely DevExpress [165], to save on implementation time. Once an event is detected, it is directly displayed and the administrator can navigate to obtain detailed information. Sensor interfaces are adapters providing communication between the MARS system and other tools. For instance, the Snort interface performs alert normalization to be compatible with the database formats. A typical deployment of these sensors is on different machines.

The incoming alerts are directed to the system memory and replicated to the storage database. The data format is mostly similar to IDMEF in addition to a few fields updated by the system during correlation analysis. For example, the aggregation process assigns a master alert for each aggregated group, the MasterID field will be updated each time the algorithm executes. Appendix II shows the graphical interface of the MARS server and client with some attack graph examples.

5.4 Real-time and near real-time implementation

As discussed in Chapter 2, there are two main algorithms for building an attack graph for alert correlation 1) scenario graph algorithms that require complete descriptions of all potential combinations of attacks. These approaches suit real-time application, though any missing information or different variations of scenarios received will cause the correlation function to fail; 2) the other algorithms, of which ours is one, are based on an attack type graph such as approaches to model cause and effect conditions, vulnerabilities and host information. These approaches are promising as they are more dynamic and tolerant of missing descriptions. However, they are mainly implemented in offline designs because of high computational requirements. Some efforts have been made to implement real-time correlation but most of them rely on a time sliding window, which consequently renders the system vulnerable to “slow-and-low” or alert flooding attacks.

Typically, event-driven applications are designed using either relational databases or real-time messaging systems [166]. The former approaches are cost-effective and provide deep analysis through data history without deadline constraints, but they mostly operate in an offline fashion. The latter approaches are capable of functioning in real-

time processing but experience difficulties in keeping up with inherent complexities in correlation systems.

The main advantage of real-time implementation is the instant detection of attack scenarios and a potentially rapid reaction. The disadvantage of these systems is the processing of alert streams, which requires the in-memory storage maintenance of a large amount of states. In addition, acquired alert data need to be stable for long enough so as to obtain an accurate correlation. And that requires a maximum time sliding window that is definite even though it is considered large enough. On the other hand, offline implementation lacks fast response, which could potentially lead to undetected scenarios in the right time to avoid system corruption. However, the usage of a database allows for the storage of a huge amount of data, reliable data reduction, dynamic updates, and definitely more chance of a comprehensive analysis to achieve higher accuracy.

Hence, we have made a trade-off between responsiveness and reliability by relaxing the real-time constraints and restricting the offline requirements. We have defined a small constant t seconds, e.g. 10 seconds, as the required interval in which to wait until a very small batch of alerts arrive to be processed. The system executes its functionalities on the database periodically and this offers a more thorough analysis. This near real-time mechanism is to avoid performance problems and the limited available data in real-time systems.

The proposed mechanism is based on the following assumptions:

1- Even though IDSs generate thousands of alerts per day, the serious attack activities are slow-and-low. Persistent attackers will not cause a large noise so as avoid notice by the operators. The conducted steps are spaced out over long intervals of days or even weeks. Sometimes, in order to defeat time sliding

windows in real-time systems, attackers flood the system with huge amounts of alerts hiding the real attack. Besides, all techniques used to prevent this type of behaviour perform a memory erasure, which certainly causes data loss.

2- Typically, for the analysis systems to be more accurate, they must be fed by a maximum amount of information. Database facilities provide a large amount of stored data.

3- The incremental aggregation mechanism and the database maintenance to ignore any redundant and already used data have minimised the performance penalty. Every time the system processes a small group of alerts, these are only analyzed using the results of the previous stages, without processing all the information in the database.

4- Alert correlation systems are implemented as complementary functions to the IDS. Hence, the delay between an alert being reported by the IDS and its analysis by the alert correlation system must be small. Extending this delay with a small time slot in order to obtain a precise view will not affect the whole process.

5- Alert correlation is not a single task, but a multi-stage functionality: correlation, alert aggregation, event aggregation and graph reduction. This complexity requires flexible memory and data storage.

We have implemented a near real-time operation for the proposed alert correlation system using a very small time for periodic database access. This will not affect the system performance, as shown in the next chapter.

5.5 Implicit correlation

Implicit correlation of alerts is used when data analysis brings out some mapping between alerts which are indirectly correlated. This approach is mainly based on

observing groups of alerts and extracts implicit correlation between them. Other works focus on creating an extra layer of relationships between capabilities which can extend the algorithm complexity. The multilayer attack classification mechanism refers each alert to different layers of attack categories. The alerts share the same generic attack class are correlated implicitly to bridge the gap which can be caused by missing alerts. However, the increase in number of the graph edges can be reduced using graph reduction techniques.

The implicit correlation is also used in our implementation for performance purposes. The aggregation mechanism is used to eliminate the number of redundant alerts as well as to reduce the search complexity. The representative alert is treated on behalf of the other aggregated alerts in the database.

The *requires/provides* model [121] is primarily based on semi-explicit correlation between logical attack conditions, unlike scenario-based approaches where logical and temporal relations are hardcoded explicitly. Thus, the semi-explicit mechanism is extended to consider implicit correlations between attacks. Relaxing the attack definitions from specific to general provides the facility for implicit correlation, as shown in Chapter 4. Moreover, we assume that attack conditions propagate from an attack to others that classified in the same intrusion category. As discussed in the previous chapter, graph reduction techniques are not intended to drop information; instead, they are intended to make use of implicit connections.

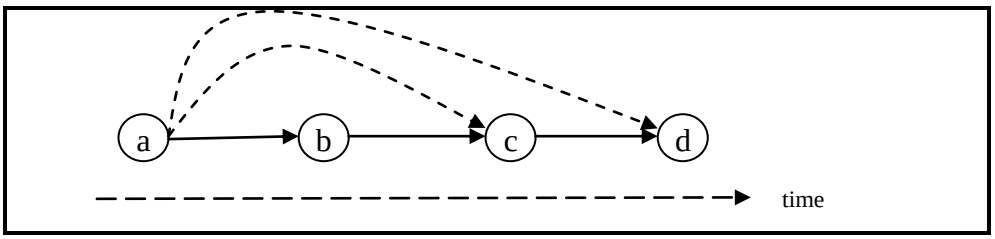

Figure 5.5 Example of implicit correlation.

Consider the four alerts shown in Figure 5.5, which are related to the same scenario detected in a temporal order. Alert a has implicit logical connections with alerts c and d. There is no need to create a link between c, d and a, as the attack conditions propagate from a to c, d and through to b. However, if c is missed, a link is created between d and b. Section 4.4 of Chapter 4 shows how the knowledge representation mechanism produces such a relationship. Most generic capabilities used to create a causal relation between two alerts are not supposed to be correlated directly in the same way as b and d. Moreover, the aggregation algorithm provides alert representatives for implicit correlations, and hence produces less computational complexity. In [33], alerts are classified first, hence more generic attack descriptions are produced, and then if two classes are correlated, all their elements can be correlated. This mechanism can increase the false positive rate and consume the system's resources. The capabilities themselves are classified to provide this generalization concept.

5.6 Conclusion

This chapter describes the development and the implementation details of our MARS framework. A system comprising several components has been developed to provide an evaluation for the proposed algorithms in chapter 4. The MARS architecture has been presented involving the MARS server, the MARS client and the administrator interactive tools. The system has been implemented to work as a near real time system providing a thoroughly analysis and at the same time responding with a short latency. We have also discussed our method of the implicit correlation to keep the process of the alerts to the minimum.

CHAPTER 6: EXPERIMENTS AND EVALUATION

6.1 Introduction

The implementation and design of our framework have been illustrated in the previous chapter. To test and validate our approach, we have implemented various experiments in a variety of test situations. A test-bench has been set up to examine system response and to measure the accuracy and performance of the algorithms and their underlying modules. A series of experiments have been conducted starting with DARPA2000, a benchmark for testing alert correlation approaches. Then a dataset collected from a controlled environment is used to measure system functionality. We have demonstrated the system's detection accuracy using two of the most common attacks in cyber crimes: SQLIA and Botnet attacks. And finally, a performance evaluation of our system in high speed networks has been conducted.

6.2 Evaluation methodology

It has been discussed in Chapter 3 the evaluation of IDS issues. The evaluation of alert correlation systems shares the same difficulties, as these are complement systems to IDS.

- *Detection Coverage:* Alert correlation systems rely entirely on the coverage detection capability of IDSs. Incomplete signature descriptions will lead to undetected activities. In addition, coverage of knowledge-base information affects the correlation process. Hence, it is essential to have attack signatures and correlation rules to achieve maximum coverage. However, correlation systems are more flexible in terms of the impact of missed descriptions. In contrast, if the IDS misses an alert, there is no way to recover it. If generalized techniques are considered, they usually result in a high rate of false positives in

IDSs. However, in alert correlation systems, generalized techniques maximize the coverage of the correlation process with minimum impact of false positive rates.

- *False positives:* An essential requirement of alert correlation systems is to reduce false positive rates, as these affect IDS function. The false positives generated by alert correlation are considered inconsistent and will be ignored. The impact of this rate is less, as the construction of attack scenarios requires only real persistent attacks. Some extra edges will be attached to the resulting graph due to the broad knowledge description. However, the level of consistency of these relations is low and only few nodes are involved along the attack path.
- *Detection Accuracy:* This measure indicates the detection rate of the true attack. It is required to expand the detection accuracy rate while false positive rates are kept to a minimum.
- *Performance in high speed networks:* The separation of the alert correlation system from the IDS distributes the required processing power and offers higher performance. Some measures are considered in this respect, such as CPU usage and memory requirements. In typical deployment, the functions that have to be performed by the correlation system require less than the IDS itself if the IDS software implementation is considered. In real-time systems, the memory limitation is the main concern, though near real-time implementation provides higher performance.

The typical IDS evaluation methodology [62, 132, 167, 168] is to run the system under test against a labelled dataset. The results obtained are analysed and evaluation measures are calculated. In most evaluation approaches, the aim is to accomplish a high rate of accuracy (soundness) and broader coverage (completeness). Soundness is the

capability of distinguishing between legitimate and attack traffic. Completeness is the ability to achieve maximum detection coverage to include attacks and their variations. In an alert correlation context, accuracy is measured using the system's ability to provide a precise recognition of causal relationships between alerts. The completeness criterion is determined by assessing the maximum correlation rate that the system can achieve. In other words, the aim is to detect diversity of complex multi-stage attacks with a minimum level of false positives rates.

In this respect, four types of criteria are considered to evaluate our alert correlation system:

- ✓ Functional characteristics: to evaluate the basic functionality to perform the correlation objective.
- ✓ Accuracy and completeness characteristics: to evaluate the quality of the correlation function.
- ✓ Reduction characteristics: to evaluate the system's capability of eliminating data redundancies.
- ✓ Performance characteristics: to evaluate system efficiency, including system capacity under different traffic conditions and resource consumption (CPU, memory, disk utilization).

6.3 Evaluation metrics

In signal processing, Receiver Operating Characteristics (ROC) [169] curves are in use to assess the quality of a receiver. Similarly, ROC curves have been used to evaluate IDSs, such as [170, 171]. The typical metrics used to illustrate IDS assessment are detection rate (the average of detected actions to the observable ones) and the false positive rate (*1-detection rate*). For signature-based IDSs as non-parametric IDSs, a

single point is used to represent the ROC instead of a curve. Furthermore, IDS measures are non-binary, in contrast with signal processing metrics. For this reason, more generic metrics are considered to assess our correlation system, which are inspired from Information Retrieval (IR) systems [172]. The four measures used are:

- ✓ True positives (TP): denotes the correctly correlated alerts.
- ✓ True negatives (TN): denotes the correctly uncorrelated alerts.
- ✓ False positives (FP): denotes the incorrectly correlated alerts.
- ✓ False negatives (FN): denotes the incorrectly uncorrelated alerts.

In IR, the confusion matrix is used to measure *precision* and *recall* rate. In an alert correlation context, *precision* is used to measure the soundness of the results and *recall* is used to measure the completeness of the results. Figure 6.1 shows the measurement terms applied to the correlation problem.

True Positives (TP) ***Correctly correlated alert rates***	**False Positives (FP)** ***Incorrectly correlated alert rates***
False Negatives (FN) ***Incorrectly uncorrelated alert rates***	**True Negatives (TN)** ***Correctly uncorrelated alert rates***

Figure 6.1. Confusion matrix.

The *recall* rate denotes the proportion of TP (correctly correlated alerts) to the total number of TP and FN (incorrectly uncorrelated alerts).

The true positive rate is the correctly correlated alert rate which is denoted by the r*ecall* rate, and the optimal measure is 100%.

$$Recall = \frac{TP}{TP + FN}$$

The *precision* rate denotes the proportion of TP (correctly correlated alerts) to the total number of TP and FP (incorrectly correlated alerts).

$$Precision = \frac{TP}{TP + FP}$$

Hence, the true alerts correlated by the system (assigned to be related but could be not related) are the total of TP and FP. On the other hand, the related alerts (known to be related and must be correlated) are the total of TP and FN. The optimal result is to achieve a higher *recall* rate with a higher *precision* rate, which means maximum precision and detection coverage. Figure 6.2 illustrates the relationships between the confusion matrix measurements.

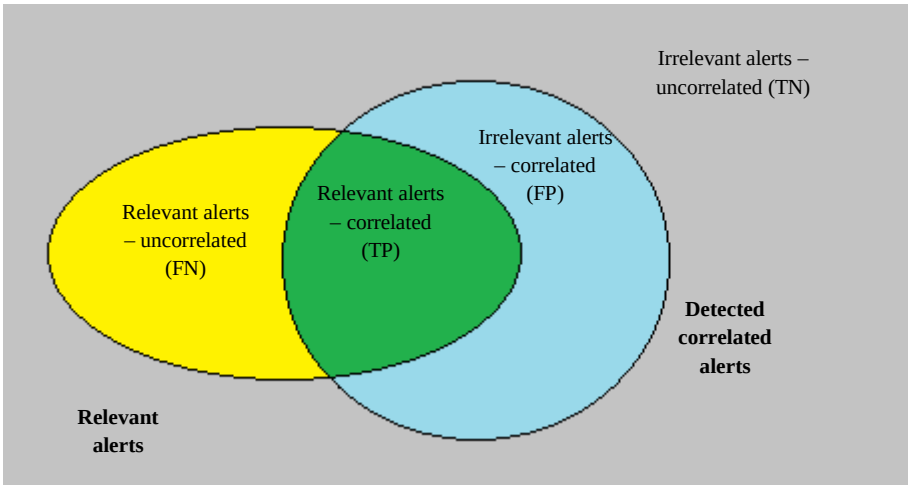

Figure 6.2. Relations between the confusion matrix measures.

The overall system accuracy can be identified by calculating the percentage of correct results (true positives and true negatives) to the total of all identified results.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

6.4 Datasets

It has been identified that the unavailability of enough benchmarking datasets is the major difficulty in evaluating IDSs in general [132]. However, there are some available

datasets have been used to evaluate alert correlation systems, such as DARPA2000 [161], Defcon [54] and honeypot datasets. However, the DARPA2000 dataset is still a reference point in the evaluation process for the comparison of results. The DARPA dataset was originally created to assess IDS sensors and is not designed for alert correlation systems. Even though it has received a high volume of criticism [22] for lack of realism of background traffic, being old and not reflecting the real attack scenarios, it is the only well-documented available dataset. The Defcon dataset, a network capture of a competition for hackers, is also commonly used to assess the correlation process. However, it is different from real-world traffic because it contains a huge volume of attack traffic only and with very limited IP addresses. The offline nature of such recorded traces creates some problems: first, the sensor alerts are not included and we have to use a certain sensor to regenerate the actual alerts, which may be different from others based on the sensor coverage. Second, the verification process is typically obtained from the status of the target at the attack time, and that has to be done manually if using capture files. Furthermore, most of these traces are synthetically created and lack a mix of the normal and anomalous traffic existing in real-life traffic.

On the other hand, the real traces recorded from real-life networks lack necessary *ground truth*. And the attack traffic in these data does not contain enough activities to represent successful multi-stage attacks [173]. In the main, datasets can be collected using five different methods:

1- A purely attack dataset with no background traffic, which is very simple to produce and is only used for basic validation of detection functionalities.

2- A dataset consisting of real background traffic obtained from production networks and synthetic attacks, which is similar to real-life traffic to some

extent. However, it is not fully controlled, has privacy concerns and is not for public use.

3- A dataset similar to 2- above but where the background traffic is sanitised to provide semi-real life traffic. However, traffic data sanitation is a cumbersome and error-prone task.

4- An entirely pure real dataset with real background traffic and real attacks captured from a production network environment. This method requires comprehensive analysis and data labelling, which is difficult, in addition to privacy concerns and being unrestrained dataset. Moreover, collected attacks are not only insufficient but require lengthy observation, which makes analysis difficult.

5- A dataset with both synthetic attacks and background traffic. The main advantage of this method is that the test environment is totally controlled and there is no potential for non-identified variables. Consequently, the results attained are more reliable and accurate. The drawbacks of this mechanism are that it is very costly because various pieces of hardware and software as well as services have to be installed, and the fact that it naturally does not reflect real-life traffic.

Our evaluation methodology is to use different datasets as follows:

- Datasets traces from .pcap files using the same timestamp for comparison purposes.
- Datasets obtained from a controlled setup to simulate real-life traffic.

6.5 DARPA 2000 datasets

DARPA 2000 datasets, including LLDDOS 1.0 and LLDDOS 2.0 [55], are often used

to evaluate IDSs and alert correlation systems. They consist of two multi-stage attack scenarios to launch Distributed Denial of Service attacks (DDoS). The evaluation goal is to test the effectiveness of our approach to recognize attack scenarios, to correctly correlate the alerts, and to minimize the false positives. This experiment is carried out mainly for functional testing to see how the system reconstructs attack stages. A reduction test is also studied in this respect; however, the background traffic in this dataset is limited. We have used these datasets for their available ground truth to assess our correlation approach and to compare our results with those of other researchers. These datasets do not contain the actual alerts from the IDS sensors, and hence we have generated them using a Snort sensor. The resulting alerts can be slightly different from others, but the generalized steps are similar.

Both DARPA 2000 datasets contains attacks conducted in stages. The attacker first probes the target system to identify the live machines, then tries to breaks into the system, then installs the DDoS software, and finally launches the DDoS attack to an off-side network. The difference between the two datasets is that the LLDDOS2.0 includes more sophisticated stages and stealthy attacks. In LLDDOS 2.0 a HINFO query has been used instead of ICMP PING for scanning live hosts. We have tested our methods on both, using the traces of the DMZ and the INSIDE network. We have used a player [174] to replay the *.pcap* files using the same delay between packets in the original traces. Snort 2.8.3, with maximum coverage configuration, has been used to generate elementary alerts that are saved in an MSSQL database connected to the MARS engine.

6.5.1 Dataset description

To evaluate the basic functionality of our systems, we replay the individual *.pcap* files corresponding to each attack phase as given by the dataset documentation. These *.pcap* files are replayed in a series based on their temporal order to be analysed by the MARS

engine. The main file containing background traffic is also replayed to obtain other necessary measurements. However, different researchers use different IDS sensors or different configurations of the same sensor to generate alerts from these datasets. In addition, the actual alerts collected during the simulation in the DARPA2000 dataset are not recorded. For this reason, we had to present the detailed description of the received alerts. Moreover, to understand how the correlation method identifies the causal relationships between Snort's alerts, the related alerts generated by Snort during the five phases of the attack of INSIDE1.0 and INSIDE2.0 are summarized in Table 6.1. Each phase has triggered certain groups of alerts according to the attacker's activities. In addition, Table 6.2 gives some traffic statistics of the four dataset captures. As mentioned in early chapters of this thesis, the performance of any alert correlation system relies entirely on the underlying IDS performance. In other words, if the IDS sensor misses some attacks, the correlation system will consequently miss the attack. However, missed attacks can be predicted implicitly by our generalized knowledge presented in Chapter 4.

In **Phase 1**, the attacker performed *ICMP PING* from a single outside IP address (202.77.162.213) to multiple class C subnets to discover live hosts in the target network and 10 hosts were live. In our knowledge-base, the alerts *ICMP PING* and *ICMP Echo Replay* have no pre-conditions, but they have a post-condition of *Disclosure of a Live Host.* Therefore, when MARS detects these two alerts, it creates a potential relation edge with corresponding attributes. And any other alert has a pre-condition of *Disclosure of a Live Host* targeting the same IP address considering the time constraints; it will therefore be correlated to the first detected alert.

Table 6.1 Description of the attack.

Phase	Dataset	Alert name	#
Phase 1: Probing	INSIDE1.0	ICMP PING ICMP Echo Reply	20 20
	INSIDE2.0	No alerts	
Phase 2: Service mapping	INSIDE1.0	RPC portmap sadmind request UDP RPC portmap Solaris sadmind port query udp request RPC sadmind UDP PING ICMP Destination Unreachable Port Unreachable	76 76 3 72
	INSIDE2.0	RPC portmap sadmind request UDP RPC portmap Solaris sadmind port query udp request RPC sadmind query with root credentials attempt UDP RPC sadmind UDP NETMGT_PROC_SERVICE CLIENT_DOMAIN overflow attempt	2 4 2 2 2
Phase 3: Break-ins	INSIDE1.0	RPC portmap Solaris sadmind port query udp request RPC portmap sadmind request UDP RPC sadmind UDP NETMGT_PROC_SERVICE CLIENT_DOMAIN overflow attempt RPC sadmind query with root credentials attempt UDP INFO TELNET Access	28 14 14 14 4
	INSIDE2.0	No alerts	
Phase 4: Installation of mainstream software	INSIDE1.0	RSERVICES rsh root	8
	INSIDE2.0	The same as in Phase 2 with different source IP address	
Phase 5: Launching DDoS attack	INSIDE1.0	SNMP AgentX/tcp request BAD-TRAFFIC tcp port 0 traffic SNMP trap tcp SNMP request tcp	4 3 1 1
	INSIDE2.0	ICMP Destination Unreachable Port Unreachable	1

Table 6.2 DARPA2000 dataset statistics.

Dataset		INSIDE 1.0	DMZ 1.0	INSIDE 2.0	DMZ 2.0
# Snort Alerts		369	1262	25	12
#Alerts types		15	14	6	6
Protocol distribution	TCP	9%	5%	9%	17%
	UDP	61%	33%	91%	83%
	ICMP	30%	62%	0%	0%
# Src IP addresses		16	20	3	2
# Dest IP addresses		22	769	3	2

In **Phase 2**, the hosts that are identified they are live, they are port mapped to determine the running services. The attacker looks for a *sadmind* daemon running on Solaris live hosts. Three hosts are running a *sadmind* service (172.16.115.20, 172.16.112.50, and 172.16.112.10), which are potential targets, creating a number of alerts by Snort.

According to our knowledge-base, these alerts have a pre-condition of *Disclosure of a Live Host* capability and a post-condition of *Disclosure of running service* capability sharing the same destination IP addresses of the alerts detected in Phase 1 and with later timestamps. Hence, a correlation has been detected between these alerts and the two alerts reported in Phase 1. Other activities associated with the discovery of the port number connected to the *sadmind* service generate other alerts, which also are correlated.

In **Phase 3**, the attacker has already gained the knowledge of the running *sadmind* daemons, and vulnerabilities trials are performed in order to break into the system. Remote-to-root attempts against each identified host are executed to cause buffer overflow attacks. The detected alerts are shown in Table 6.1, including overflow attempts and queries with root credentials. These alerts are consequences of the disclosure of a running service and associated port number. The impacts of these alerts on the target machine are potential *System Access*, *Remote Access* and *Admin Access*. The attacker then tries to verify the level of achieved access using *TELNET ACCESS*, creating the corresponding response from the IDS sensor.

In **Phase 4**, a *.rhosts* file is installed in */tmp* directory in order to start up the *mstream-sol* software on the victim hosts.

In **Phase 5**, the attacker manually launches the DDoS using *TELNET login* on the victim machines running the master daemons of the *mstream* software. These activities are not detected by the IDS sensor; however, they can be detected as bad traffic behaviour if Snort is configured to do so. All related packets have spoofed source IP addresses using random TCP ports on the victim machines.

LLDDOS2.0 is also a DDoS multi-stage attack using a stealthy behaviour to avoid detection. Instead of *ICMP PING*, a *DNS HINFO* query is used to find out which hosts are Solaris. *HINFO* records contain information regarding running OSs, only *sadmind* query is performed, and those hosts are reported as Solaris. As shown in Table 6.1, there are undetected activities in Phase 1. In Phase 2, the same alerts detected in INSIDE1.0 are detected, including port mapping and overflow attempts. The victim machine (172.16.115.20) is broken into and is used to launch the attack. For this reason, it is not necessary to consider DMZ capture. In LLDDOS1.0, the attack is accomplished from outside the network, so DMZ and INSIDE captures should be considered.

We have conducted our test to evaluate the functionality of MARS by replaying the individual .pcap files of each phase. Then, the whole traffic capture is used in order to evaluate the reduction functionality.

6.5.2 Functional test

The five *.pcap* files are replayed in a temporal order using the same timestamps to simulate the real attack. An interval of approximately three hours of traffic is analyzed by Snort using our test-bench, and alerts are sent to the MARS server where the database is located. The detected events evolve over time instead of by batch analysis. The results obtained are shown in Table 6.3, both with and without the alert verification mechanism.

Table 6.3 Functional test results, DE: detected events, RA: related alerts, CRBAG: Correlation Rate Before Aggregation, CRAAG: Correlation Rate After Aggregation.

Dataset	**Alert verification disabled**				**Alert verification enabled**			
	DE	RA	CRBAG	CRAAG	DE	RA	CRBAG	CRAAG
INSIDE1.0	10	325	2164	84	3	91	661	48
DMZ1.0	18	984	1464	138	3	91	439	52
INSIDE2.0	2	16	24	16	2	16	16	16
DMZ2.0	1	8	12	8	1	8	8	12

The number of detected events are shown in Figure 6.3, and it is clear that this number is reduced using the verification techniques. Only three events associated with successful attacks are identified. For the datasets INSIDE2.0 and DMZ2.0, there is no reduction because the attacker has only targeted Solaris systems.

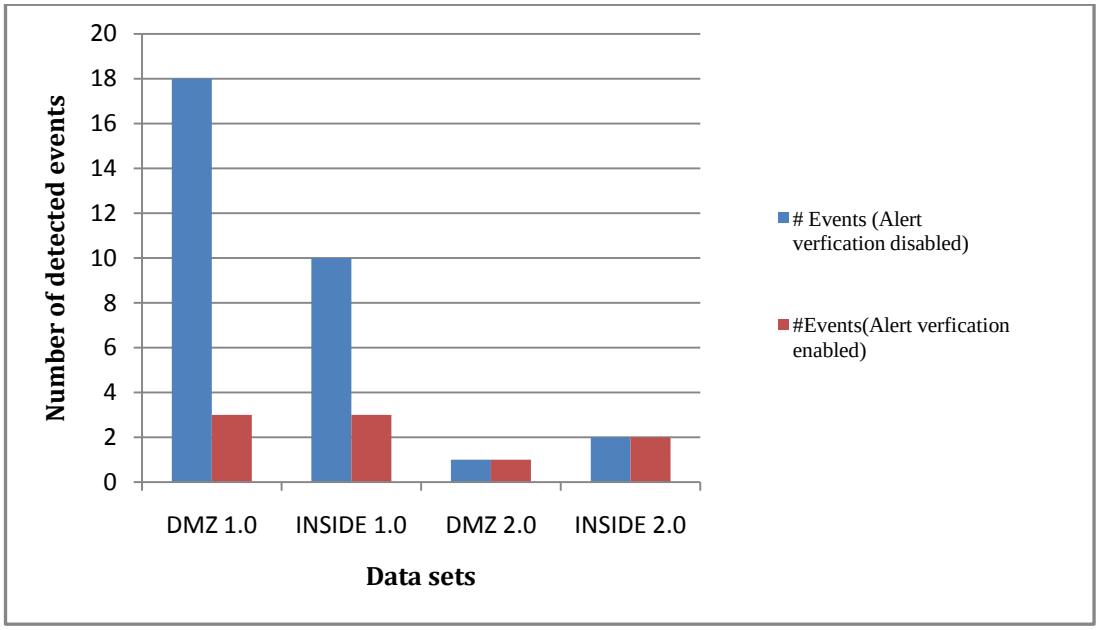

Figure.6.3 Detected events in the functional test.

- INSIDE1.0: the system has detected a total of 10 events evolved over time. All events are related to the actual attack; however, only three events are related to successful attacks associated with the IP addresses (172.16.112.10, 172.16.112.50, and 172.16.115.20). This is determined by the vulnerability model to verify the potential of successful attacks. The three events are similar in stages as the attacker (202.77.162.213) has performed the same sequence of attack attempts, as shown in Figure 6.4.

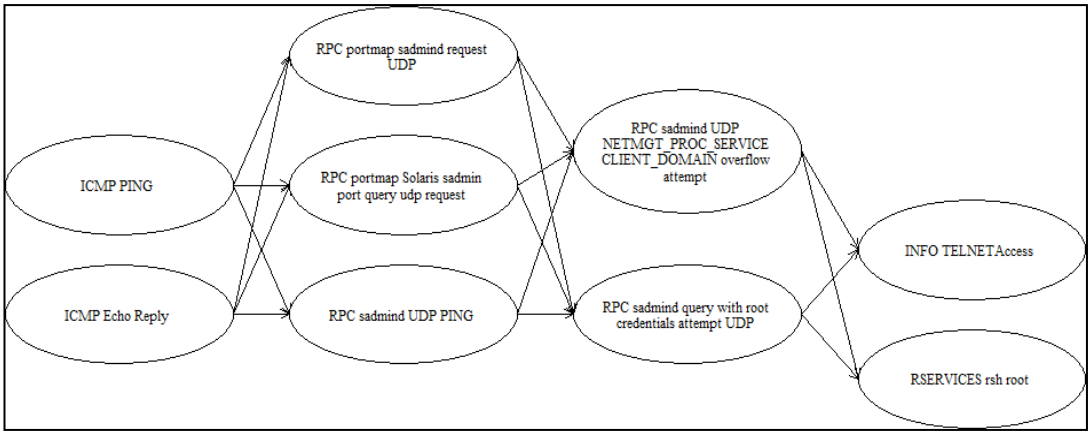

Figure 6.4. Attack graph of the three detected events.

It should be noted that the alerts displayed in Figure 6.4 are aggregated alerts based on our aggregation algorithm discussed in Chapter 4. Phase 5 is not detected because Snort itself does not detect any activity explicitly said to be a DDoS attack. However, if Snort decoders are enabled, 502 alerts classified as bad traffic are detected, which could be a sign of DDoS behaviour. It is not reliable to correlate these alerts, as spoofed IP addresses are used and this will have a negative impact on system performance, producing a high volume of false positives in real life. Detecting the installation of suspicious software in a protected network is more important than detecting actual DDoS activities because the attack source will be under control in its initial stages.

Certain other events are detected if the alert verification is disabled reflecting unsuccessful attacks, such as the host (172.16.115.87) shown in Figure 6.5. The attacker has carried out the scanning stage and then a *sadmind* service discovery has been performed; however, the target host has not responded and the attack attempt is ended after two stages. This behaviour is considered a medium-priority event by MARS, as the target host is not running a *sadmind* service.

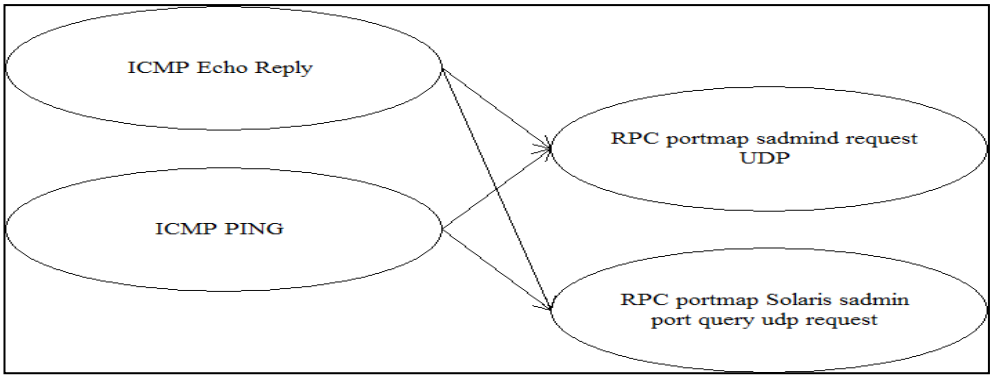

Figure 6.5. Attack graph of non-critical events(INSIDE1.0) detected by MARS.

- **INSIDE2.0**: The nature of the LLDDOS2.0 multi-stage attack is to be stealthy, reducing the noise amount over the target network. MARS has detected two events associated with the hosts (172.16.115.20, and 172.16.112.50) and there is no scanning stage. Only eight alerts (five aggregated alerts) are involved in each event and three out of five stages are detected, as shown in Figure 6.6.

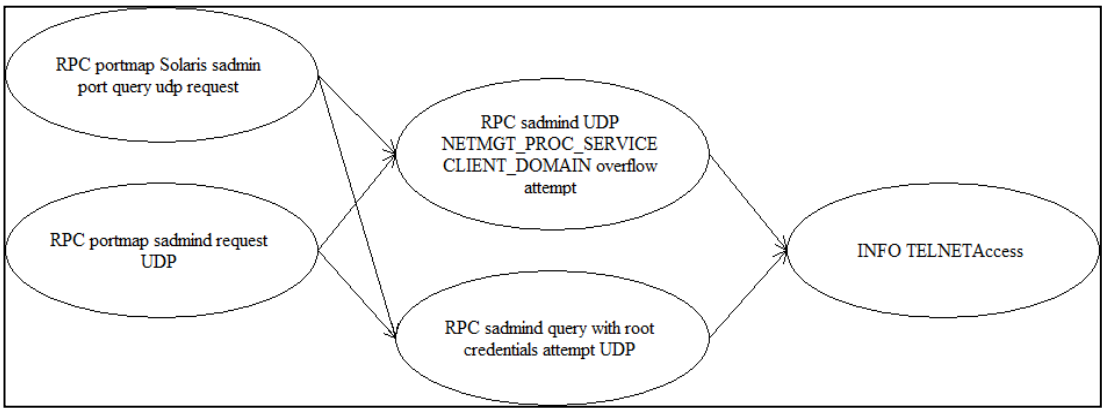

Figure 6.6. Attack graph of the events detected in INSIDE.2.0.

The alert verification technique does not reduce the number of detected events, as the attacker only targets Solaris hosts where a *sadmind* daemon is running.

6.5.3 Accuracy reduction evaluation

In our initial work [175], certain experiments with DARPA 2000 datasets have been performed to comparatively validate our approach. The goal of this initial evaluation was to test the effectiveness of MARS in recognizing attack scenarios, correctly correlating alerts, and minimizing false positives. In addition to our system, the system

developed by [35] (TIAA) is used for comparative evaluation. Table 6.4 shows the results obtained, and a few main points can be summarized:

- Snort has not detected the behaviour of launching the DDoS attack itself; however, the sequence of the attack has been detected.

- Certain different alerts are related to the same attack, such as *sadmind* daemon attempts. However, these alerts are not ignored because the correlation system should identify such cases.

- In the second scenario of the attack, a large amount of behaviour went undetected due to the stealthy nature of the attack. However, the correlation system has to recognize the security situation by discovering the causal relationships between alerts.

It is clear from Table 6.4 that the rate of both false positives and false negatives in LLDDOS1.0 have been improved. However, the unsatisfactory results from the experiment of the second dataset LLDDOS2.0 are similar to the TIAA system due to the inability of Snort to detect all the attack activities. To measure the effectiveness of the proposed system, the false positives and the false negatives are calculated according to the definition of the confusion matrix presented in section 6.2.

Table 6.4 Comparative results to evaluate MARS effectiveness.

		LLDDOS1.0		LLDDOS2.0	
		DMZ	Inside	DMZ	Inside
Elementary alerts		3684	720	1214	199
Related alerts		1262	369	12	25
# correlated alerts (relevant)		206	182	8	7
# correlated alerts (detected)	TIAA	275	235	13	11
	MARS	223	198	11	11
# correctly correlated alerts (TP)	TIAA	174	155	3	6
	MARS	184	165	3	6
# Incorrectly uncorrelated alerts (FN)	TIAA	32	27	5	1
	MARS	22	17	5	1
False positive rate	TIAA	25.1%	22.5%	38.5%	36.4%
	MARS	8.25%	8.1%	27.3%	36.4%
False negative rate	TIAA	15.5%	14.8%	62.5%	14.3%
	MARS	10.7%	7.23%	62.5%	14.3%

The graph in Figure 6.7 shows the main evaluation metrics, which are the false positives and false negatives of both systems. It is clear that MARS achieved better performance in all dataset traces. However, the high level of false positives for DMZ2.0 traces is justifiable because most of the attack activities have been conducted from inside the local network. The outside attacker has broken into a vulnerable host and continued attacking other hosts locally.

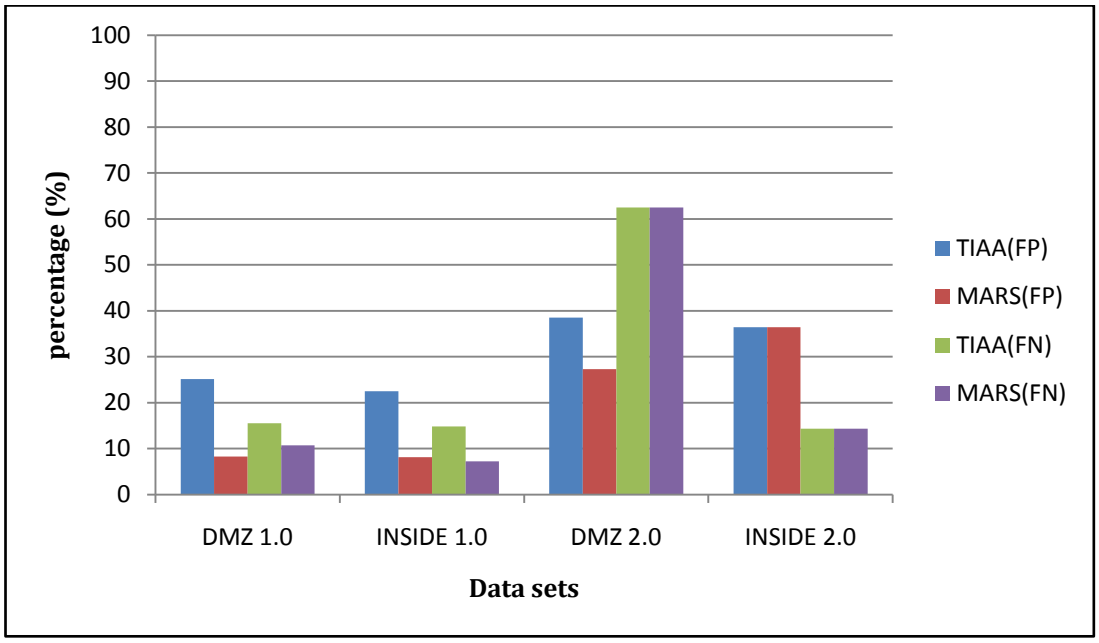

Figure.6.7 The main evaluation metrics of MARS and TIAA.

In the other previous works, the accuracy measurements are mainly based on the number of correlated alerts regardless of how many correlation instances are associated with the same alert. For example, there is no difference between two situations when alert *a* is correlated with alert *b*, and the same alert *a* is correlated three times with *b*, *c* and *d*. This case in our evaluation is considered to provide a more accurate assessment. To achieve more accuracy, the number of relationships between alerts is considered instead of only the number of correlated alerts. Providing a ground truth for a dataset based on correlation instances is not an easy task and can be very difficult for huge

datasets. Our system has been evaluated using this mechanism on the DARPA dataset because of extensive data description available. In addition, the dataset has been comprehensively analyzed, assigning each single packet to its associated behaviour. This has been achieved by a manual effort in addition to automatic analysis using certain tools, such as BASE [176]. For each alert received from the Snort sensor, all possible correlation chances are computed and are categorized according to whether or not they are related to the main scenario. It should be noted that this process focused on the four stages of the DDoS attack detected by the sensor.

The reduction functionality is vital in alert correlation systems in order to measure the system's capability to minimize alert redundancy and false alarm ratios. For this reason, experiments have been implemented on the LLDDOS traffic *.pcap* files using whole recorded packets including background traffic. This methodology gives us a broader evaluation context beyond detection functionality. Accuracy metrics are calculated to determine *recall, precision* and *accuracy* rates. Our analysis results for the DARPA dataset are summarized in Table 6.5

		Alert verification disabled				Alert verification enabled			
		LLDOS1.0		LLDOS2.0		LLDOS1.0		LLDOS2.0	
		DMZ	Inside	DMZ	Inside	DMZ	Inside	DMZ	Inside
# elementary alerts		3684	720	1214	199	3684	720	1214	199
# related alerts		1262	369	12	25	131	171	12	16
Correlation rate	# relevant correlations	1849	2915	61	91	530	623	27	33
	# detected correlations	1788	2959	69	96	528	628	26	37
	TP	1636	2731	42	67	513	613	18	28
	FP	152	228	27	29	15	15	8	4
	FN	213	184	19	14	17	10	9	2
	TN	340	322	16	23	58	37	8	28
	Recall rate (%)	88.4%	93.7%	60.9%	73.6%	96.8%	98.4%	60.3%	87.5%
	Precision rate (%)	91.5%	92.3%	68.9%	82.7%	97.2%	97.6%	68.3%	93.3%
	Accuracy	84.4%	88.1%	55.8%	67.7%	94.7%	96.3%	63%	90.3%
Correlations with aggregation		177	156	22	65	66	103	12	16
# detected events		25	17	3	6	3	3	1	1
# aggregated alerts		135	114	17	37	36	50	8	14
Reduction rate		96.3%	84.2%	98.6%	81.4%	99.1%	93.1%	99.3%	92.9%

Table 6.5 Evaluation results of the DARPA datasets – accuracy test.

Figures 6.8 to 6.11 illustrate the key results presented in Table 6.5. Our proposed system has achieved high levels of accuracy among the datasets in LLDDOS1.0, and acceptable levels in LLDDOS2.0. The only low accuracy rate recorded is from the analysis of the DMZ2.0 dataset, and of which we are aware because the actual attack was performed inside the network. The vulnerability model to verify the importance of alerts is also showing a considerable improvement. This is apparent from the number of detected events in each dataset. For instance, in DMZ1.0, the number of events has been reduced from 25 events to only 3 related events. The overall accuracy rates are higher if alert verification is used and satisfactory for other tests. In addition, the volume of alert information has been significantly reduced, achieving more than a 90% reduction rate in most test cases.

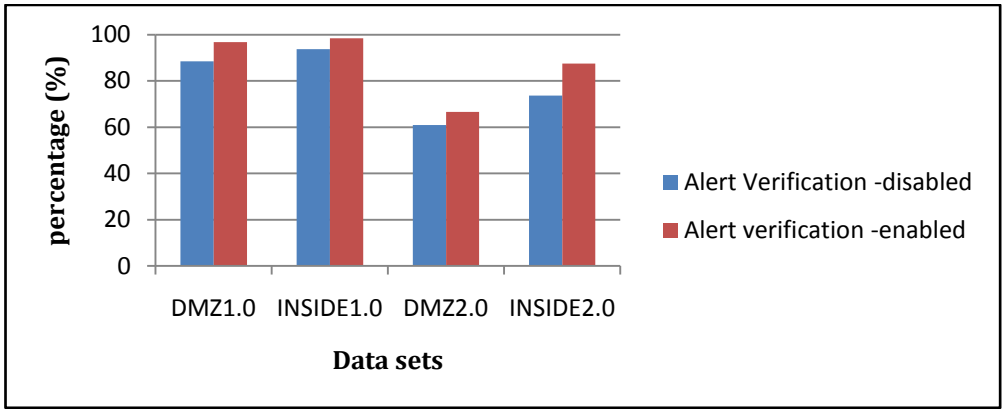

Figure 6.8 Recall rate (%) of the DARPA dataset.

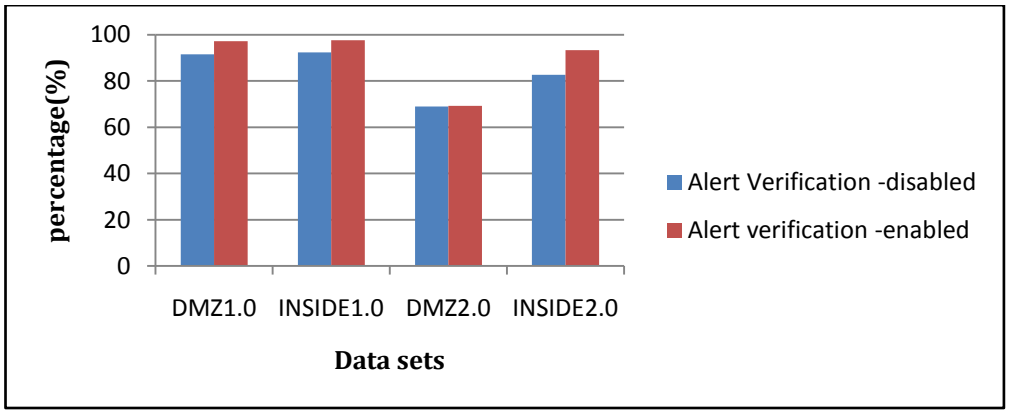

Figure 6.9 Precision rate (%) of the DARPA dataset.

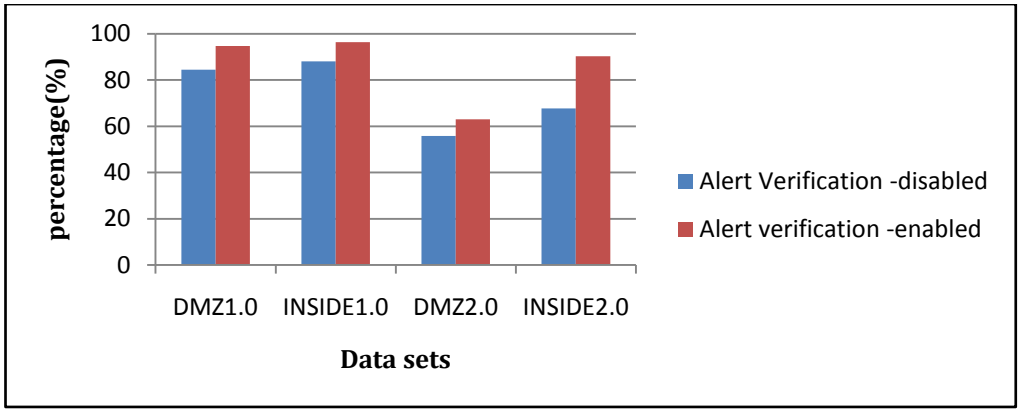

Figure 6.10 Overall accuracy rate (%) of the DARPA dataset.

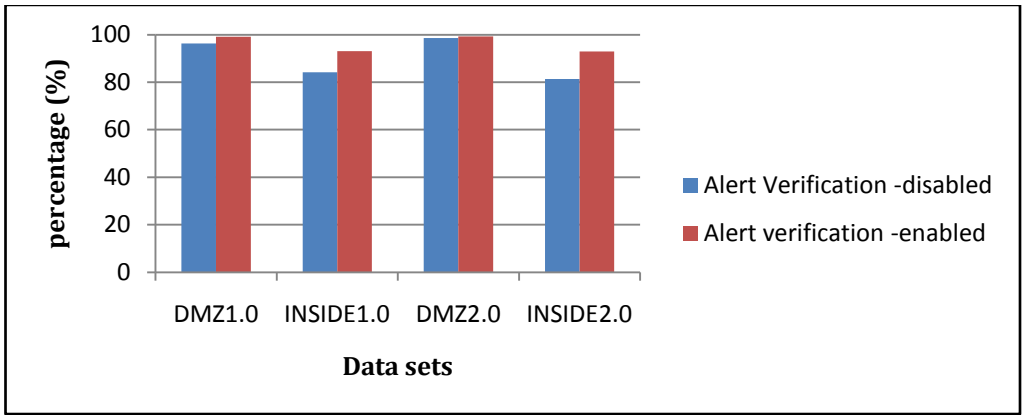

Figure 6.11 Alert reduction rate (%) of DARPA dataset.

6.6 Real-life experiments in a controlled setup

To mitigate the problems in the previous datasets, a controlled network setup environment has been used to simulate real attack stages. The proposed system has been evaluated in a real high-speed network composed of actual and virtualised machines connected via switch. Figure 6.12 shows the experimental setup, where multiple machines are designated to communicate with services installed on our servers. This is to reflect the normal background traffic that can be found in real-world scenarios. The false positives generating machine is to add some noise by creating different isolated attacks to different machines. The attacking machine is used to carry out the real attack. The Snort sensor is connected to the switch's monitoring port in order to analyse the passing traffic.

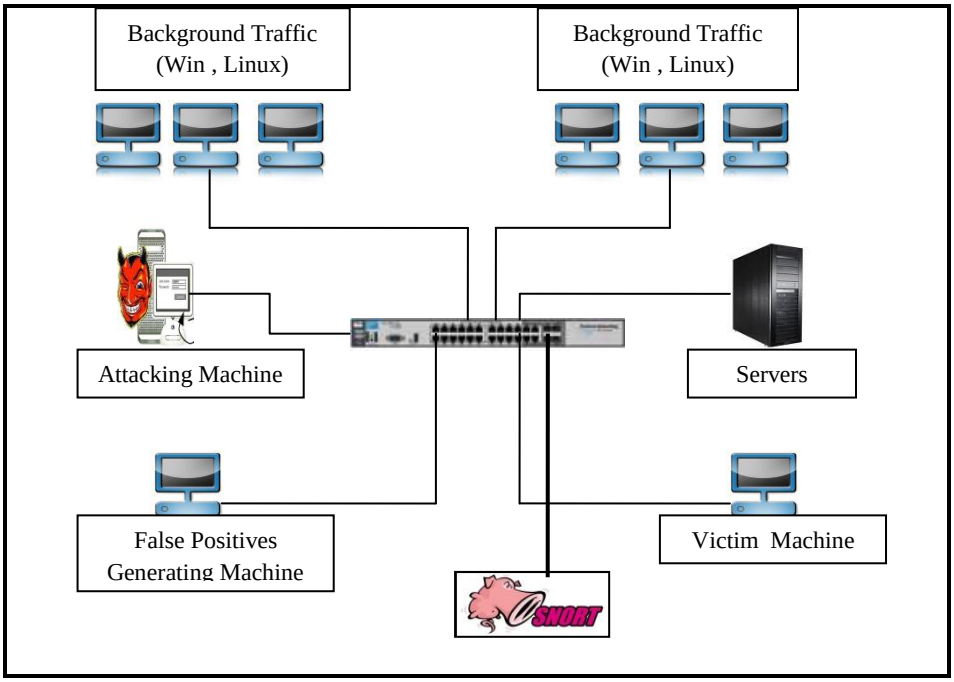

Figure 6.12. Test bench.

To produce a reliable evaluation test, the following conditions are considered:

1- The traffic should contain malicious attack traffic as well as false positives, because in a typical case the IDS sensor generates a high amount of false alarms. In addition, some isolated real attacks should be injected to assess the correlation system's capability.

2- The environment should be controlled and every single action should be documented. In this respect, the traffic of each machine has been recorded individually and reanalysed it by Snort after the experiment. This is in order to determine the original source of each generated alert. The total alert repository is compared by individual alert containers in order to obtain a ground truth for our evaluation.

3- The truth file is generated manually by matching individual alerts to the attack stages. For the sake of simplicity and accuracy, each attack stage is performed using different IP addresses, which are then changed to the original values. Each group of alerts is assigned to the corresponding attack stage.

We have evaluated our approaches using two common Internet attacks as case studies: Botnet and SQLIA.

6.6.1 Botnet attack experiment

The Botnet attack is a multi-stage and coordinated process, and to detect such activity we need to obtain the whole picture of the attacker behaviour. Network-based and host-based IDSs can detect certain attacks based on their signatures or protocol analyses. However, detected events are treated as isolated activities and uncountable variations of Botnets are discovered every day. Attackers tend to change their fingerprints to avoid detection by IDS rules despite the fact that the general behaviours are similar. Even though the IDS misses some attacks involved in Botnet activity, the network administrator is still aware of the global view of a suspected Botnet behaviour. In addition, according to several behaviour analyses [177, 178], Botnet communications and activities are similar regardless of the common name of any used malicious software. For instance, Zeus, Kneber and Bredolab [177] are variations of the same malicious modular Botnets.

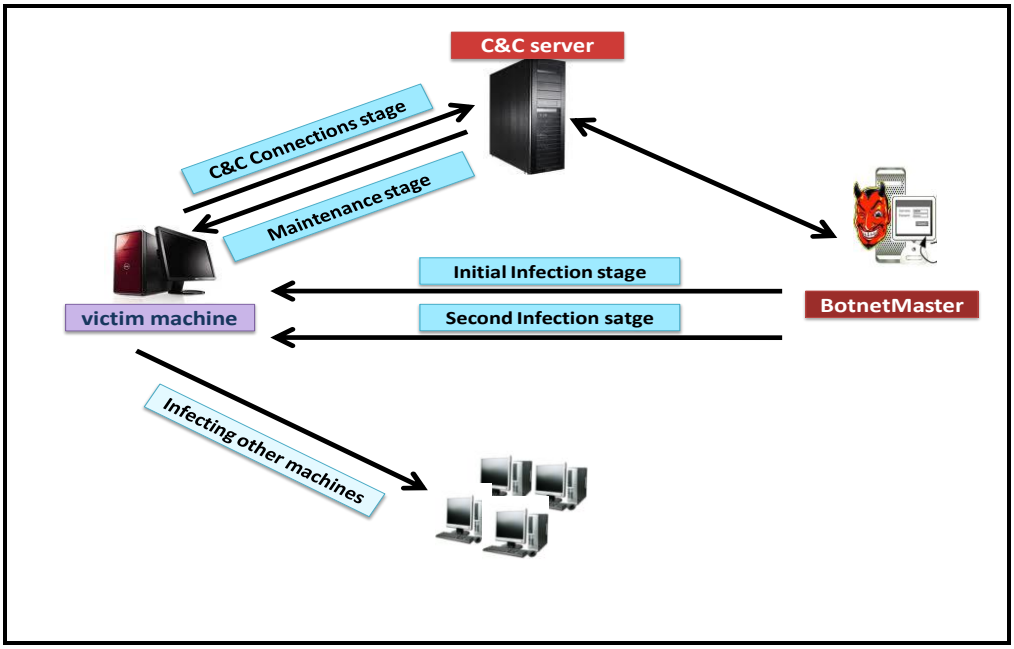

Figure 6.13. Botnet lifecycle.

In spite of the fact that different Botnets have been identified in the field of security analysis, almost all follow similar steps, which are known as Botnet lifecycles. These sequences are shown in Figure 6.13 and summarized as follows:

1) *Initial infection stage*: This stage involves scanning for systems running vulnerable services or responding to backdoors.
2) *Second infection stage*: Remote malicious code is loaded and software is installed in the target machine using one or more available attack vectors. The infected system is ordered to download the actual Botnet software from a dedicated Bot server. Then, the code is executed and the machine becomes a Botnet member.
3) *Connection to the C&C stage*: The infected machine connects to the attacker and receives commands to be configured and updated using C&C channels over IRC or HTTP. In this stage, the actual Botnet activities begin.
4) *Attacking other machines stage*: Scanning activities are maintained to discover un-patched and vulnerable systems to launch further possible infections.
5) *Maintenance stage*: Depending upon the capabilities of the target machine, the attacker commands the Botnet members to download binaries, to connect to another C&C server and to become involved in attacking other victims. The attacker also has to be certain that all members can be reached using the Fast Flux DNS technique [179] to hide malicious code deliveries under all dynamic network conditions.

Zeus [177, 180] Botnet is one of the emerging modular Botnets reflecting the darkness of cyber crime world, first identified in 2007. It is also known as banking crimeware and was motivated initially to steal banking credentials and account information. Some of its abilities include stealing data submitted by HTTP forms, emails and FTP account information, stealthy injection of HTML on the fly, and all redirection activities to trap victims. It is a software package with GUI and its builder is responsible for creating all

necessary files such as executable files, PHP files and SQL templates in a straight forward manner.

An older version of Zeus (as the new versions are sold by licence) has been installed on one of our machines in our lab and on an isolated network. We have followed the typical real-life scenario in simulating the traffic communications between the Bot master and the victim machines. The simulated network is monitored by Snort and the MARS engine. Snort is configured with all rules enabled including: VRT [23], bleeding-Edge [181], Community, and Emerging Threat rules (ET) [182].

6.6.2 The basic functionality test for Botnet experiment

In this section, a simulation of the Zeus Botnet attack has been used to test the detection efficiency of the proposed approach. We have pursued a Botnet scenario as occurs in real networks, as described later in this section. Initially, the network traffic used consisted of attack traffic only in order to assess the effectiveness of MARS. The transmitted traffic has been recorded individually in *.pcap* files for further analysis. Based on the typical Botnet scenario, each attacking action is assigned to its corresponding stage. The attack steps and their related alerts are shown as follows:

1) The attacker starts scanning, looking for vulnerable systems to exploit or to install a backdoor in the target machine. In this scenario, the attacker will use a new identified application flaw, which is CVE-2010-0188 [39], Adobe Reader in versions earlier than 9.3.1. An embedded executable code launch command can be used to infect the target machine. Metasploit [26] is used to perform this job by copying a malformed malicious PDF document to the victim machine. Snort has triggered two signatures related to scanning activity and three other signatures in connection to Shellcode and CVE-2010-0188 vulnerability. As shown in Figure 6.14, the five alarms are correlated in a

sequence. This scenario is not necessarily Botnet activity, because it could be any other attempt to obtain system access.

sid: 1394 SHELLCODE x86 inc ecx NOOP

sid: 16490 SPECIFIC-THREATS Adobe Reader malformed TIFF remote code execution attempt

sid: 15013 WEB-MISC Adobe Portable Document Format file download attempt

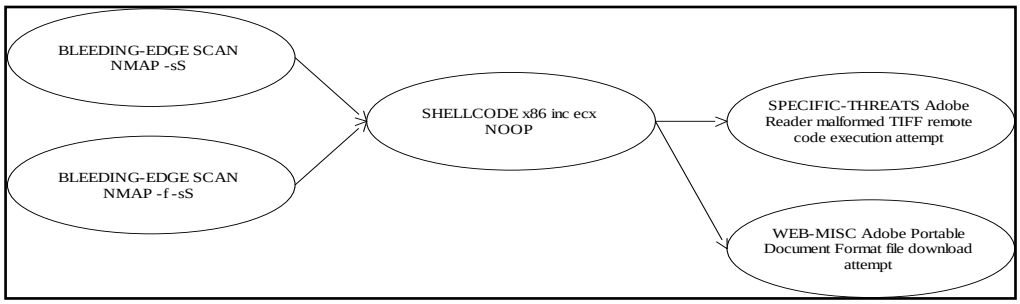

Figure 6.14. First attack stage.

2) The target host is infected and starts to connect to the C&C server to download binaries and configuration files. An HTTP GET request is sent to the C&C server to obtain encrypted configuration files. While these files are encrypted and their names and since the URLs are random, it is very difficult for Snort and all other signature-based IDS to detect such files. However, an alarm has been triggered in this stage recognizing the name of the configuration file. These signatures have been added to Snort VRT rules in version 2.8.6.1 in July 2010 [23].

sid: 2008100 ET TROJAN PRG/ Zeus InfoStealer Trojan Config Download

sid:16912 BLACKLIST URI request for known malicious URI - net/cfg2.bin

The previous signatures are part of a group of signatures to block certain suspicious URI requests containing malicious websites tracked by Zeus Tracker [180].

3) Followed by the configuration files, an HTTP POST request was sent to the same C&C server used in the second stage to fetch PHP files, and again the data in POST

request is encrypted. Snort fired an alarm similar to the alarms in the second stage but with different URIs.

sid:16929 BLACKLIST URI request for known malicious URI - gate.php?guid=

4) Despite the fact that the previous two steps can be performed without a Snort response using some obfuscation techniques, this stage can be identified. The server response for the last step contains some recognized behaviour, which is the string *Content-Type:text/html*, and the actual data are not in HTML or other legitimate formats. Actually, there is a signature in Snort that can catch this piece of traffic, which is *sid:16460* [23], but it is deleted due to false positive concerns, as this case may exist in normal traffic. Therefore if we have a system that recognizes false positives generated by Snort, and this is the case with the MARS system, this alert will be ignored if it is not involved in a real attack scenario. For this reason, the 16460 rule is enabled to provide more information, and in case of an isolated false alarm, it will not contribute to the attack picture. In addition, Snort has triggered other alerts based on ET rules that identified some small binary downloads, which are suspicious behaviours that need to be noticed. The sequence of correlated and aggregated alerts involved in this stage and the previous two stages are shown in Figure 6.15.

sid:16460 WEB-MISC text/html content-type without HTML-possible malware C&C

sid:11192 POLICY download of executable content

sid:2003179ET POLICY exe download without User Agent

sid:2007671 ET POLICY Binary Download Smaller than 1 MB Likely Hostile

sid:2009033 ET POLICY Suspicious Executable (PE under 128)

sid: 2000419 ET POLICY PE EXE or DLL Windows file download

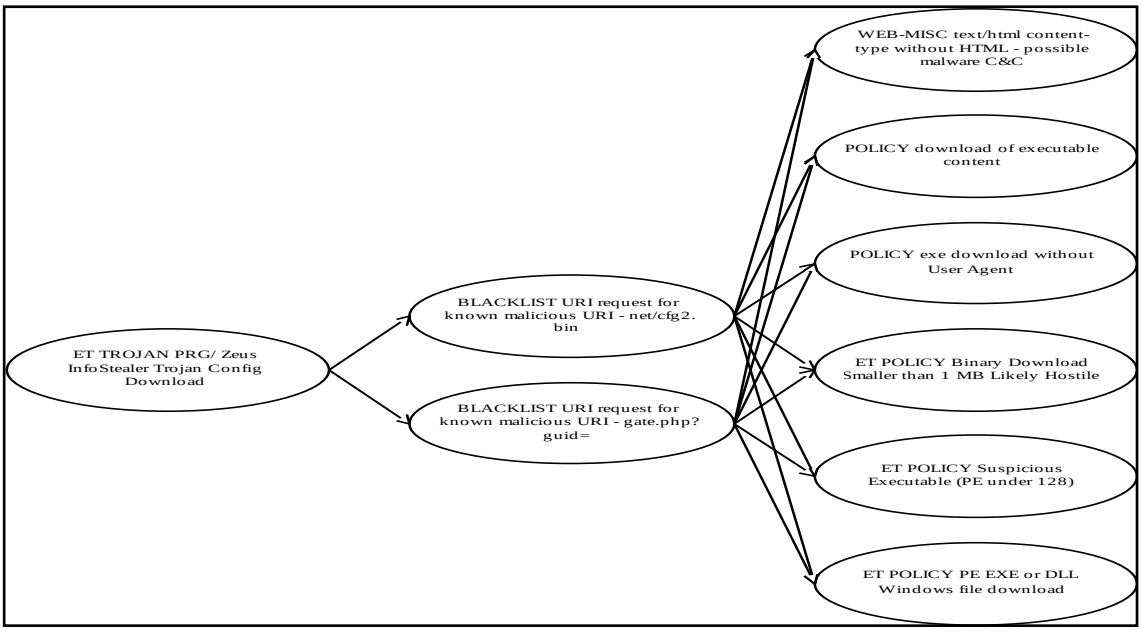

Figure 6.15. The second, third and fourth attack stages.

5) The last stage involves maintenance and update by downloading further binaries. In addition, the infected machine participates in fast scanning and visiting malicious websites that can be detected by policy rules. And on certain occasions, the infected machine sends large numbers of DNS requests experiencing query failures or redirection, which are very obvious signs of a Botnet attack. This part of the attack scenario is shown in Figure 6.16, and the whole attack graph is shown in Figure 6.17.

sid: 2009028 ET MALWARE 404 Response with an EXE Attached - Likely Malware Drop

sid: 2009885 ET SCAN Unusually Fast 404 Error Messages (Page Not Found), Possible Web Application Scan/Directory Guessing Attack

sid: 2011085 ET POLICY HTTP Redirect to IPv4 Address

Figure 6.16. The fifth attack stage.

It should be noted that these stages can be extended to perform the main purpose of the infected machines, such as DDoS, spam and the distribution of malware. These activities will also be included in the attack map if originated from the same machine.

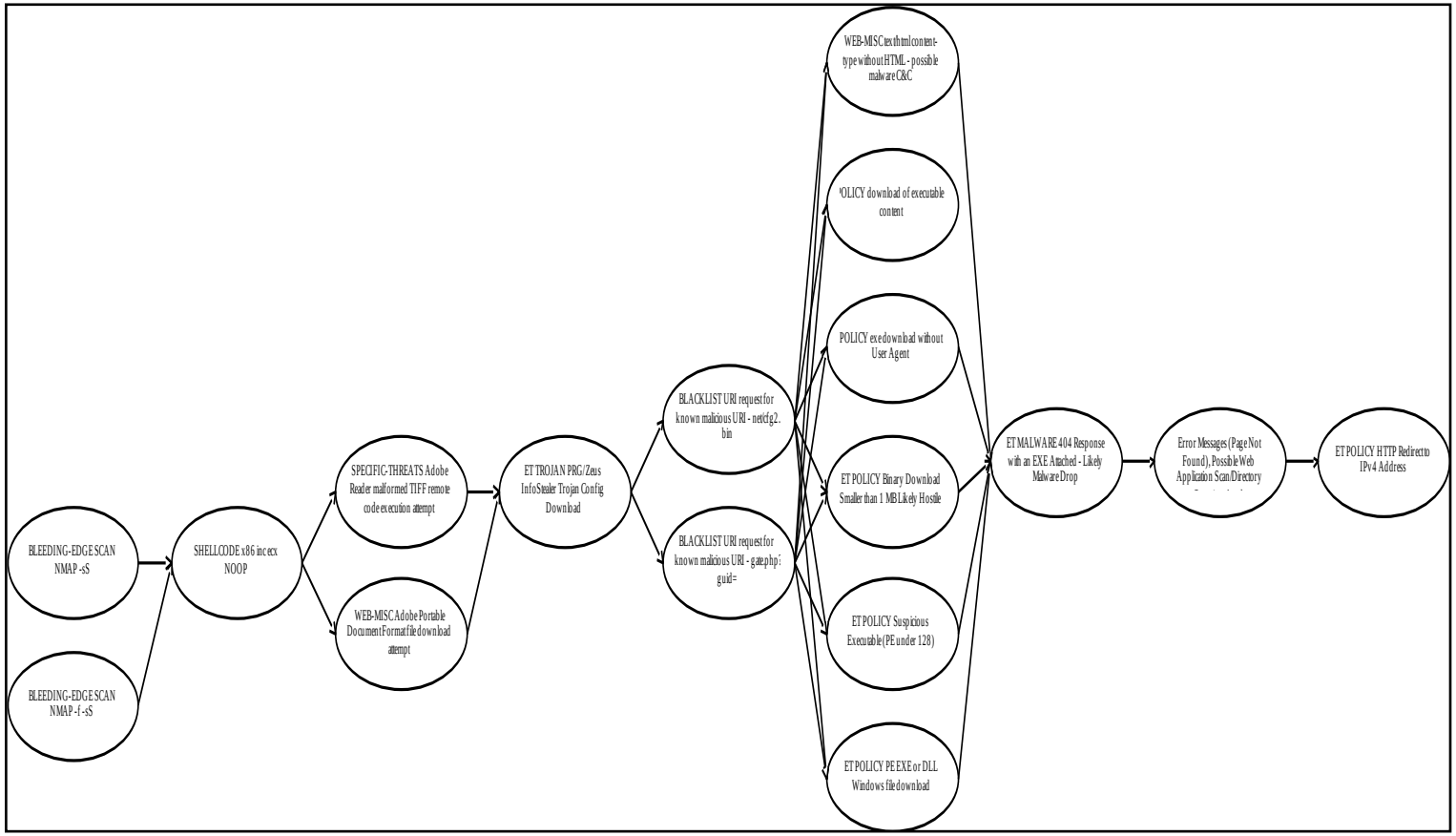

Figure 6.17. Graph of the extracted Botnet scenario.

6.6.3 SQL injection attack (SQLIA) experiment

An SQLIA is also a multi-stage and coordinated process, and to detect such activities we need to correlate the attacker's actions. Some of these attack actions are detected by IDSs based on available signatures. However, unlimited SQLIA variations are discovered every day, in addition to the use of evasion techniques to deceive IDSs. Moreover, some of the generic attack symptoms can be detected and are typically considered isolated alerts. Hence, we need a correlation system to process different activities over time in order to provide a global view of the attacker's intention.

The typical SQLIA scenario is summarized as follows:

1. Application fingerprinting via input validation [183]: in this stage the attacker tries to discover any vulnerable entry into the Web application using certain basic techniques. This involves testing a Web page form its fields, query strings in URL REQUEST and POST, or crafted values in cookies. A combination of strings such as ′ , ″ ,), #, --, etc., can be sued to generate possible application errors. The attacker has to perform this stage to fingerprint the application and the database, otherwise it will be unclear which SQLI technique should be used. This stage can be a combination of scanning and port mapping activities using some available fuzzing tools.

2. Database fingerprinting: to gather information about the application and the database incorporated in the previous stage. Analysis of different responses through error messages is used to choose the appropriate method of injection, and this is based on the type of the target database. Then database column numbers are discovered and which ones are vulnerable. In this respect, different databases use different syntax.

3. Attack stage: by exploiting the detected vulnerable columns to obtain extra information, such as database version, server name, user table name, etc. A typical injection technique is to use SELECT UNION to craft query statements in URL requests.

4. Information disclosure: includes extracting data in user and password tables. It is based on the available privilege levels gained by the attacker. Data modifications can be performed, such as adding new user accounts and making deletions and updates.

5. Advanced attack: to interact with the operating system in order to achieve full control over the target system. Therefore, this is the most dangerous action that can be exploited by such attack. This stage involves uploading files, such as shared objects for Linux and Dynamic Link Library (DLL) for Windows. User-defined functions supported by SQL databases can be used for more interaction with the OS through direct command execution. Furthermore, the attacker can add some user accounts and local groups to the OS.

We consider SQLIA as a multi-stage attack conducted by an attacker to compromise a target system. PHP Web application vulnerabilities are exploited to gain access to the MySQL database and to obtain table names, column names and stored data. Consequently, the attacker acquires administrator privileges to upload malicious files to control the Web server hosting the target application. We have followed the typical scenario in real-life, simulating the traffic communications between the attacker and the victim machine. The simulated network is monitored by Snort and the MARS engine where Snort is configured with all the rules.

6.6.4 The basic functional test of the SQLIA experiment

In this section, simulating SQLIA has been used to test the detection accuracy of the proposed approach. We have pursued the SQLIA scenario as occurs in real networks, as described later in this section. Network traffic has been recorded in a *.pcap* file for further analysis. We have used a similar technique as the one implemented in the last experiment using attack traffic only, and then the experiment will be repeated using a mix of real and synthetic traffic. The attack steps are as follow:

(1)The attacker starts to perform scanning and port mapping, looking for running services; Snort has triggered two signatures related to scanning.

sid: 2000537 BLEEDING-EDGE SCAN NMAP -sS

sid:2000545 BLEEDING-EDGE SCAN NMAP -f -sS

(2)Discovery of vulnerabilities (basic SQLI techniques)

The attacker will initially use the basic techniques such as the symbols: ' , * , and " to determine if the target website is vulnerable to SQLI. Also some other strings such as 1 = 1 , '1' = '1'))/*, or 1=1--, or "a"="a", can be injected.

And if an error is displayed on the target website, it means that the site is vulnerable, e.g.: Warning: mysql_result(): supplied argument is not a valid MySQL result resource

Snort generates certain alerts related to this stage:

sid:1000303 WebAttack PHPInjection test \' detected

sid:1000304 WebAttack PHPInjection test 1=1 or 1=2

Figure 6.18 shows the evolving attack events generated by the MARS system based on Snort alerts.

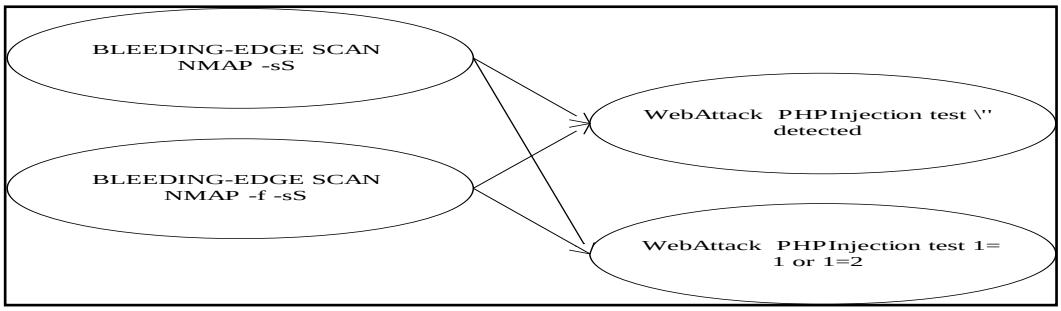

Figure 6.18. The first triggering event.

(3)Discovery of the number of columns in the target database: The attacker uses ORDER BY to determine the total number of columns in the database. For instance, the statement includes ORDER BY 1 to request the page to display the first column if no error is displayed. This number is increased and decreased until the exact number of columns is identified.

(4)Discovery of the vulnerable columns out of the identified ones in carried out in step 2. The UNION SELECT statement is used to identify which column is vulnerable that can be exploited to get access to the database, as follows:

```
UNION SELECT 1,2,3,4,5,6,7,8
```

Then all vulnerable columns are displayed; in this experiment, 4, 6 and 7 are vulnerable. Three types of signatures are generated by Snort in this stage.

```
sid:2010963 ET WEB_SERVER SELECT USER SQL Injection Attempt in URI

sid:1000302 WebAttack PHPInjection -1=select detected

sid: 1000305 WebAttack PHPInjection -union allselect
```

(5)Exploitation of the vulnerable columns to disclose the database information: In this step, the attacker discovers the database name, version and usernames by substitution of these variables in the vulnerable columns fields of the UNION SELECT statement as follows:

```
UNION SELECT 1,2,3,4,5,concat(database(),version(),user()) ,7,8
```

This statement will display the requested information on the column 6 position of the site page. This step is important because different versions of databases, MySQL in our case, have different syntaxes. Snort responds with certain alarms.

```
sid:2011042 ET WEB_SERVER MYSQL SELECT CONCAT SQL Injection Attempt
sid:2011073 ET WEB_SERVER Possible Attempt to Get SQL Server Version in URI using SELECT VERSION
sid:1000302 WebAttack PHPInjection -1=select detected
sid:1000305 WebAttack PHPInjection -union allselect
```

(6)Disclosure of table names, usernames and passwords. The attacker will use the same statement in step 5 to identify the names of the database tables, and to obtain the login names and passwords from the user table. An example of this statement is shown below:

```
UNION SELECT 1,2,3,4,5,concat(table_name,column_name,table_schema),7,8 FROM information_schema_tables
WHERE column_name LIKE %pass%
```

Snort triggers similar alarms to the previous step.

(7)File privilege server path discovery: Knowledge of file privilege levels is very important in order to read, write and upload files. An example of the statement used in this respect is:

```
UNION SELECT 1,2,3,4,5,load_file('/'),7,8 FROM information_schema.user_privilages
```

In order to upload files to the target server, it is necessary to determine the server paths, and there are different and easy techniques for this, for example:

```
UNION SELECT 1,2,3,4,5,@@datadir,7,8
```

The next action is to check the directories with write permission. Temporary directories are the best choice in this respect, such as: /temporary/ , /temp/, /images/,/cache/, …etc.

```
UNION SELECT 1,2,3,4,5,load_file('/etc/password'),7,8 INTO OUTFILE '/home/www.site.com/images/passFile.txt'/*
UNION SELECT 1,2,3,4,5,'<?system($_get["c"])?;>',7,8 INTO OUTFILE '/home/www.site.com/images/c.php'/*
```

Below are some related Snort alerts:

```
sid:2010037 ET WEB_SERVER Possible SQL Injection INTO OUTFILE Arbitrary File Write Attempt
```

sid:1020053 WebAttack PHPInjection load_file

Figure 6.19 shows the extracted attack graph for the detected events.

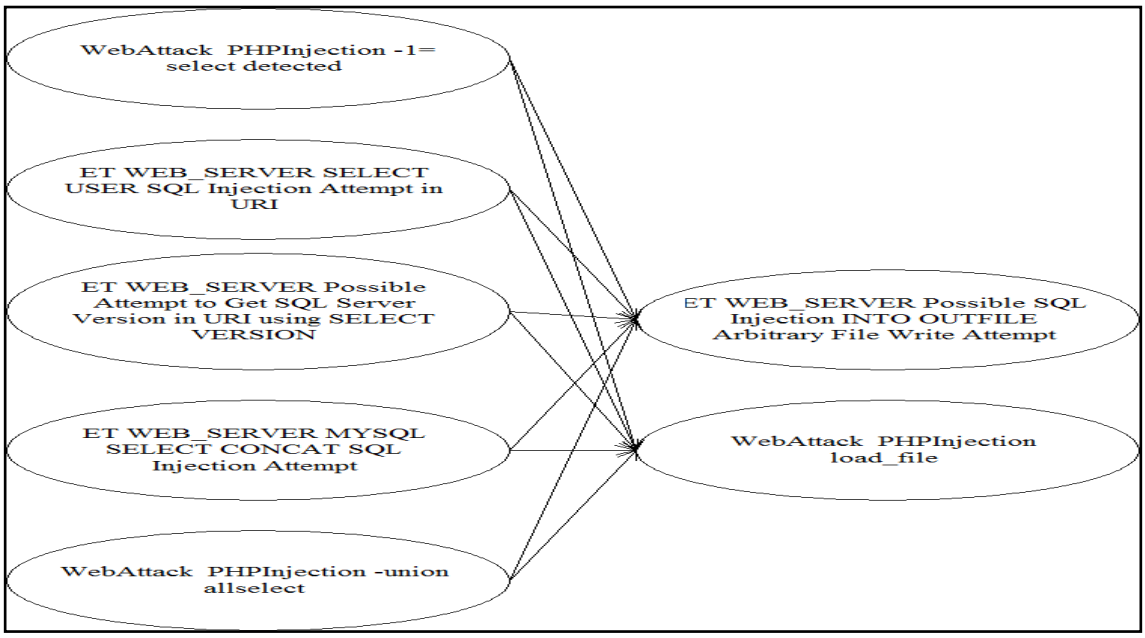

Figure 6.19. The second and third events.

(8)Advanced attack stage: The attacker can list, modify, insert and delete some or all information in the target database. In addition, files containing scripts or libraries can be uploaded to configure the server and to provide more interaction with the operating system. Snort reacts to some activities while others are not detected. However, the correlation system can support discovery of the whole behaviour.

sid:1100061 WebAttack SQLInjection QueryData Domain

sid:2006443 ET WEB Possible SQL Injection Attempt -- DELETE FROM

sid:2006444 ET WEB Possible SQL Injection Attempt -- INSERT INTO

sid:2006447 ET WEB Possible SQL Injection Attempt -- UPDATE SET

Figure 6.20 shows the whole extracted attack graph consisting of a detected SQLIA event.

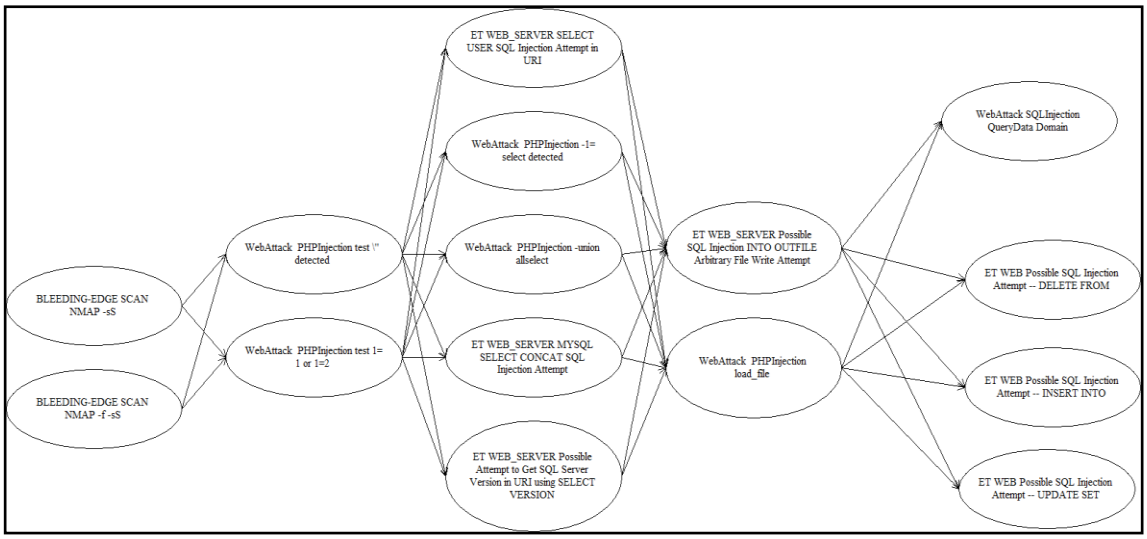

Figure 6.20. Extracted SQLIA scenario graph.

6.6.5 Accuracy and reduction evaluation

In the last sections we have evaluated the correlation functionality for detecting Botnet and SQLIA behaviour. Then, the same steps are repeated by mixing the original attack with background traffic, isolated attacks and false positives. This is to determine the *recall*, *precision* and *accuracy* characteristics of our approach simulating a real-life environment. The target machine is attacked by Metasploit, with a similar behaviour to a Botnet attack. The Nessus tool is also used for generating scanning behaviour. The background traffic contains synthetic traffic using the traffic generator [134] and real traffic using a communication with running services. The real traffic is limited compared with the synthetic one, but at least it is more reliable than pure artificial traces. It should be noted that we have intentionally avoided creating any other multi-stage attacks than our planned attack. The multi-stage attack has been performed carefully over a period of three hours to allow for intervals between steps. The generated data is labelled using source IP addresses, destination IP addresses and timestamps. We met the test requirements described earlier in this chapter to avoid errors.

Table 6.6 Accuracy and reduction evaluation for Botnet and SQLIA experiments (AVd: alert verification disabled, Ave: alert verification enabled).

		Botnet		SQLIA	
		AVd	AVe	AVd	AVd
# Snort alerts		1328		1186	
FP (Snort) (%)		76%		73%	
Correlation rate	# relevant correlations	753	492	661	91
	# detected correlations	777	496	669	96
	TP	734	489	637	67
	FP	43	7	32	29
	FN	19	3	24	14
	TN	135	112	117	23
	Recall rate (%)	97.5%	99.4%	96.4%	97.5%
	Precision rate (%)	94.5%	98.6%	95.2%	96.8%
	Accuracy	93.3%	98.4%	93.1%	95.8%
Correlations with aggregation		87	52	66	45
# detected events		8	1	11	1
# aggregated alerts		78	53	74	45
Reduction rate		94.3%	96%	93.8%	96.2%

Table 6.6 shows the results of the evaluation test for both Botnet and SQLIA experiments. The improvement in accuracy measures over DARPA datasets is due to the fact that all attack stages were detected by Snort. Only one high-priority event would be detected if alert verification is used. The other events are with low priority, and these consist mainly of scanning behaviour. In addition, in a real-world situation, multi-stage attacks are not frequent and do not cause any noise because the attacker must achieve its target in a stealthy manner. It is observed from these experiments that the system has achieved significant data reduction and only one event is detected using the alert verification component.

6.7 Performance evaluation

It has been identified in Chapter 3 that performance is a critical factor for the IDS as a real-time system. This experiment has been performed to evaluate the performance of the MARS engine as a complement component to IDSs. The performance

characteristics include resource consumption (CPU and memory usage) and processing time of each alert. The objective of this experiment is to show that the correlation engine will not affect the performance of the IDS's detection functionality. Two groups of experiments have been conducted: one for offline implementation to measure alert processing time, and the other for online implementation to assess performance under different traffic volumes.

For the offline test, Snort and MARS are tested to process a batch of a number alerts starting from 1,000 alerts to 100,000 under the same conditions. A pcap file contains 1,300,000 packets to generate 1000 alerts are read by Snort and thereafter processed by MARS system. Then the pcap file is replayed to generate alerts from 1000 – 100,000 alerts. The CPU and Memory usage reading is taken from the task manager. The test has been conducted on a Dual Quad-Core 2.0GHZ machine with RAM of 4.0 GB. Snort is configured to log to an MSSQL database the same as MARS. The database server is installed on the same machine to evaluate the worse performance case. Figure 6.21 and Figure 6.22 show the results obtained for both systems.

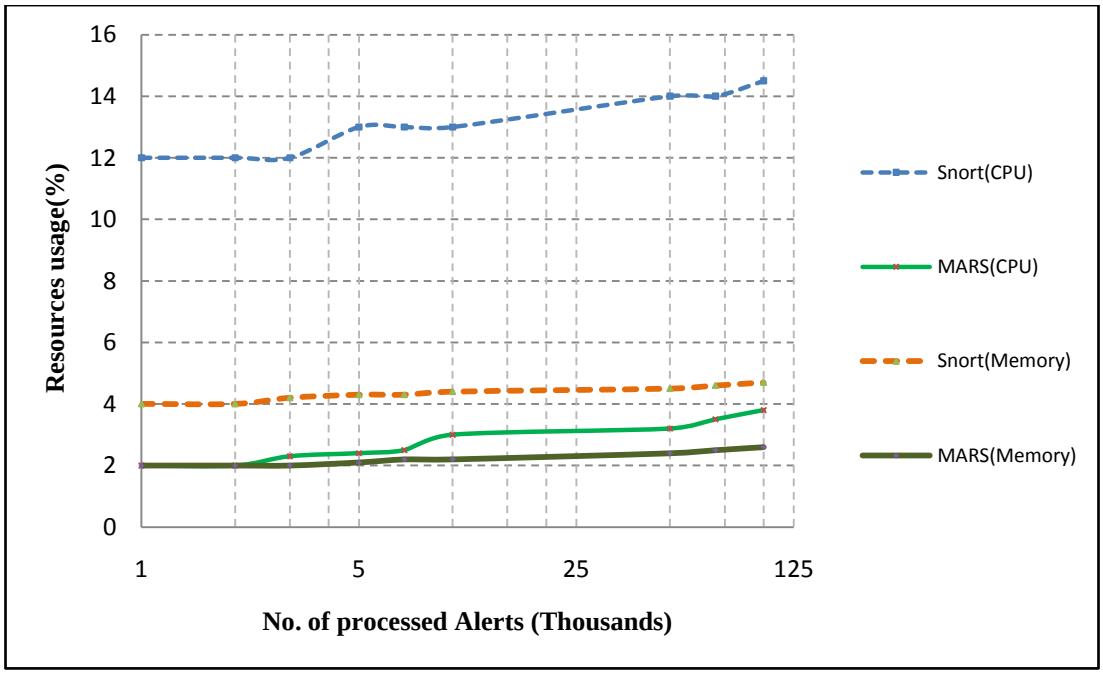

Figure 6.21. Comparison of resource usage by Snort and MARS (offline test).

These illustrate that the MARS correlation engine consumes less resources compared to the Snort system for both CPU and memory usage. The alert processing time using Snort increases proportionally with the number of alerts input. However, the MARS engine is relatively stable even with the increase in the volume of alerts. And this is explained by the fact that Snort inspects packet headers and content whereas MARS inspects alert information.

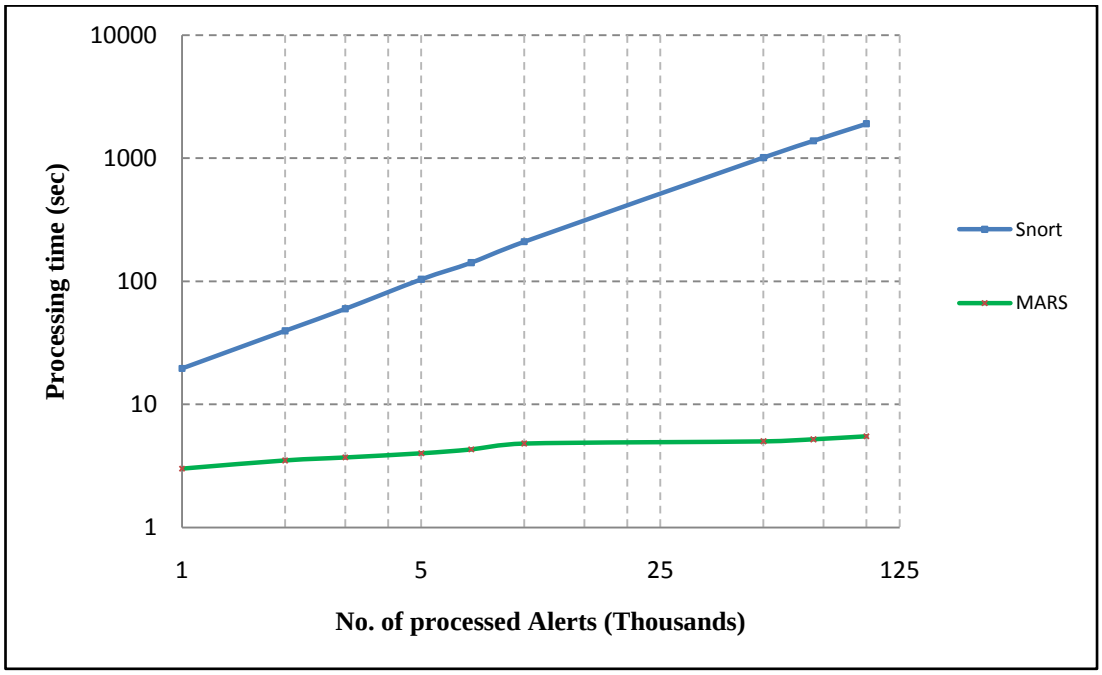

Figure 6.22. Alert processing time of Snort and MARS.

A real-time evaluation has been conducted to demonstrate that the correlation system will not affect overall performance, even with high-speed network traffic. The performance of Snort in high-speed environments has been studied intensively in Chapter 3, demonstrating the efficacy of Snort. It has been observed that Snort is capable of processing incoming traffic with speeds of less than 750MB with acceptable levels of packet dropping. Both systems have been tested with traffic ranging from 100MB to 750MB, as shown in Figure 6.23. The CPU resource demand by Snort increases dramatically with higher traffic speeds. On the other hand, the MARS engine

continues to be stable, as the required processing power is less compared to the deep packet inspection performed by Snort. Hence, alert correlation systems can be deployed without requiring considerable resources.

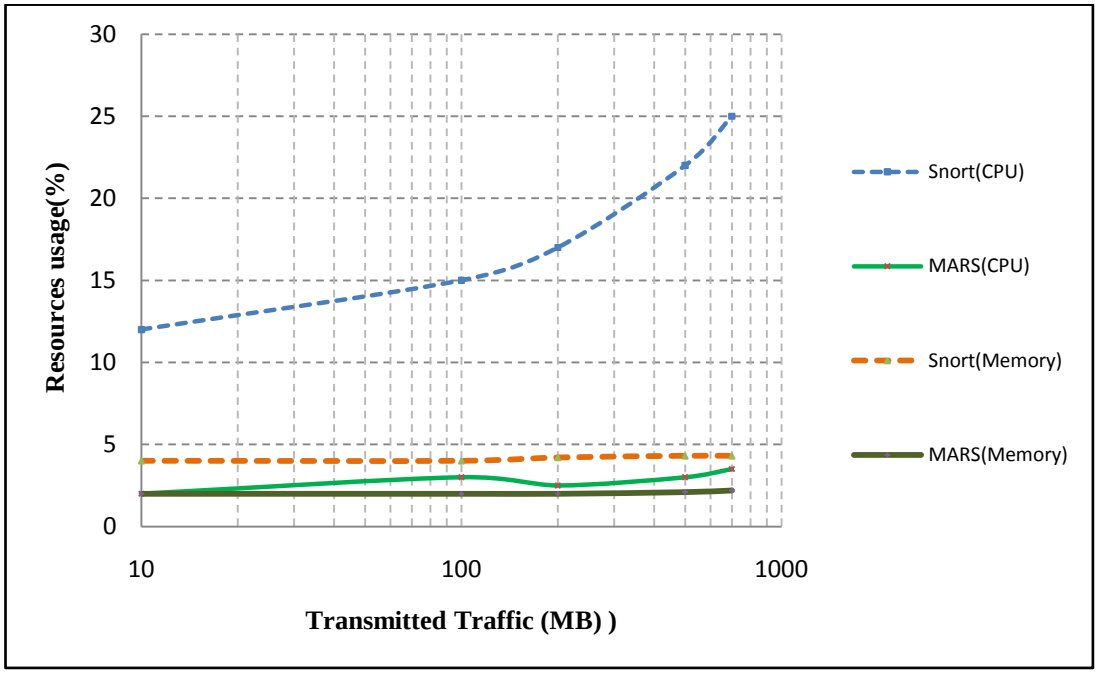

Figure 6.23. Comparison of resource usage by Snort and MARS (online test).

It has to be mentioned that is not a comparison between Snort and MARS as equivalent systems that perform the same functionalities. Snort processes all received packets and inspects them against its rules to generate alerts. MARS takes these alerts and analyses them to obtain the coordinated attacks while the number of alerts is typically less than the number of network packets. Hence, in Figure 6.22, Snort has processed about 1,300,000 packets to produce 1000 alerts and only these alerts has been processed by MARS. The goal of this experiment is to prove that MARS engine does not affect the overall performance of the NIDS system.

6.8 Conclusion

In this chapter, we have discussed the evaluation issues related to the assessment of alert correlation systems. Majority of the evaluation approaches are based on finding the

rates of false positives and false negatives. We have evaluated our system using different metrics to identify the functionality, the reduction and the accuracy rates. An experimental platform has been developed to perform different tests. The MARS tool has been tested using DARPA 2000 data set to compare our results with others. The obtained results have showed that the proposed system is capable to detect all attack instances with lesser false positive rates. Then, we have implemented a real life test using a controlled testing environment to evaluate the MARS capabilities to detect two types of current cyber attacks i.e Botnet and SQLI attacks. It has been demonstrated that our framework can applied to detect complex multi-stage attack. Botnet and SQLIA traffic have been analyzed as case studies to measure accuracy and performance of MARS tool. We have confidence that our system has achieved an improvement in relation to identification of attack plans and reduction in graph complexity. False positives have been reduced comparing with other approaches using vulnerability knowledge base. We have also evaluated the performance of the MARS system using offline and online testing. It has been demonstrated that the MARS function does not affect the system's overall performance as the resulting latency is mainly caused by the NIDS system e.g Snort.

CHAPTER 7: CONCLUSION AND AVENUES FOR FUTURE RESEARCH

7.1 Introduction

Network Intrusion detection systems (NIDSs) are gaining widespread interest as a complement to traditional preventative techniques. More critical data is migrated to online systems, which creates the need for efficient data-protection mechanisms and monitoring tools. However, the performance of NIDSs is still debatable in terms of the nature and amount of traffic to be processed, as well as detection accuracy. In this thesis, we have addressed several complex issues in the field of NIDS technology in relation to performance in high-speed networks and alert management systems.

In the initial stages of this research we implemented a comprehensive evaluation methodology to measure the capability of software-based NIDSs in keeping up with increasing network bandwidth. NIDSs, as network product systems, must exhibit the same performance requirements and traffic characteristics. Furthermore, NIDSs perform highly performance-intensive functions, such as deep packet inspection and state maintenance. We have provided in-depth NIDS performance analyses utilizing a representative real-life Gig network environment. The focus has been mainly on the performance evaluation of Snort as a de facto open-source NIDSs. The results obtained illustrate that software-based NIDSs installed on a general-purpose machine are not capable of keeping up with traffic above 750Mbs in an ideal scenario. When the NIDS becomes unable to handle packets in real-time, it starts to drop these, potentially resulting in attack patterns being injected into the protected network.

The identified packet loss problem in the NIDS performance test in the initial phase of this research has raised some considerations that have motivated the work implemented

in the rest of this thesis. We have found a proportional linear relationship between packet loss and the rate of missed attacks. The missed attack limitation can be the result of other factors, such as the absence of signatures and the use of evasion techniques. However, we have adopted a dual-solution approach to mitigate both problems irrespective of the root cause. Alert correlation systems have been widely used in network management systems to localize the fault cause and to determine the dependencies between detected events. We have proposed a reasoning alert correlation framework consisting of several integrated components to draw an attack graph. Our approach can build an overall view of the system's security status even with incomplete alert information. The outcome of the proposed framework is the minimisation of the effects of missing audit data, the reduction of the large volume of redundant alert which are mostly false positives, and the extraction of an attack behaviour summary in the form of a multi-stage attack scenario.

Received alerts are analysed and abstracted to a higher level of attack description using a generalisation mechanism. Pre- and post-conditions inspired from *requires/provides* models are applied to detect correlation characteristics. Supporting knowledge-bases are formalized in a multi-level abstraction based on attack taxonomies. Generic signature representatives in the form of attacker capabilities are constructed to obtain the relationships between elementary alerts. To achieve an effective correlation system we have used alert verification based on a vulnerability knowledge-base to suppress the generalisation methods in the generalised attack concept.

The proposed alert correlation framework has been implemented in a tool called MARS (Multi-stage Attack Recognition System) to validate our approach. A repository contains alert data, mapped IDS signatures and detected events. Knowledge-bases reside in-memory, where they are used by the detection engine. In addition, interactive

administrative tools are developed to dig in resulting events and aggregated alerts. The developed system has been evaluated by various datasets applying different realistic attack scenarios. Evaluation metrics have also been described in detail, and the results obtained have been demonstrated in various forms.

7.2 Evaluation of NIDSs in high-speed environments

The main objective of this part of our research is to deal with the shortage of information available on the evaluation of the performance of NIDSs in high-speed networks. Most previous efforts have focused on the accuracy functions of the NIDSs using moderate traffic loads. We have intended to provide a realistic evaluation methodology reflecting real-life situations. We accept as true that if some packets are not analysed by the NIDS under high traffic loads, this means that it will become vulnerable to evasions from attackers. We have elaborated our evaluation tests using extreme conditions under various scenarios. A multi-tier test methodology has been utilized to investigate the system-under-test response, starting from moderate to advanced hardware implementations. Snort performance has been evaluated on different operating system (OS) platforms using host-based and virtual configurations. The virtualisation test was motivated by its successful inception within the industry/business community.

The test-bench was established using 10 Gbps network cards and supported by a Xenon Quad-Core server and other machines with multi-processing powers. To achieve practical results, we customised the traffic to be semi-real traffic with a range of packet sizes, protocol distributions, number of connections and elapsed times. The outcome of the test can be summarized as follows:

- It can be ascertained that Snort is not suitable for all configured implementations with high volumes of traffic, e.g. above 750Mbps.

- There are no significant performance improvements, even with multi-core processing configurations. In practice Snort, being a non-multithreaded design, does not utilize the processing power provided by the hardware implementation.

- The implementation of Snort on virtualised platforms and using the current configuration is not promising. This is realistic as virtualisation has its limitations in terms of disk I/O performance.

- Packet drop caused by Snort performance efficacy results in the degradation of overall system effectiveness and opens it up to overload and evasion attacks.

7.2.1 Avenues for future research

Architectural techniques to improve Snort performance are not the main focus of our research. However, along the course of the performance evaluation, certain researches directions have cropped up that lend themselves to recommendation.

There should be a mechanism to achieve a lower rate of packet drops by utilising the available multi-core system. This can be done by performing an architectural re-design of the Snort system to scale for high volumes of traffic with minimum packets loss. A parallel concept can be employed for Snort in order to distribute the system processing. In other words, Snort could be rebuild as a multithreaded application to run multiple threads concurrently in order to utilise the processing power of multi-core systems. This will increase the packet processing capabilities of Snort, and hence result in fewer packets dropped.

Splitting traffic over multiple instances of Snort engines have been studied in some respects [14, 43]. However, these mechanisms have not been implemented and evaluated in a systematic manner. The received network traffic can be divided based on

flow level to insure that all packets related to a single flow are passed to the same engine. The main concern is the problem of the distribution mechanism, as it adds more burdens to the process. Efficient algorithms for light-weight processing techniques are required to achieve cost-effectiveness.

It has been identified in the NIDSs literature that updated datasets are a necessity in testing NIDSs, whether for performance or function evaluation. DARPA datasets are still the benchmark in spite of being old and criticised for lack of realistic background traffic. Moreover, there is a shortage in open-source traffic generators, and the available tools have some limitations and cannot be relied upon.

7.3 A reasoning framework for alert correlation

The NIDS fire alerts coressponding to individual activities that are isolated from others, leaving the prediction of incoming attacks to the adminstartor's estimation. Detection of the actual intrusion may fail due to a variety of reasons: attacks may be missed due to performance degregation, the corresponding signature not being provided, or the attacker using a new variation of the intrusive behaviour. In order to provide a remedy for missed attacks, whether caused by packet loss or the absence of signature descriptions, we have proposed a fault-tolerant solution. The proposed system gathers all information required to construct a context for understanding the attacker's behaviour. A state record of each activity is built using aggregation and intellegent correlation of detected events. Therefore, the decision is made according to a higher level of information fusion.

A reasoning framework for alert correlation has been presented consisting of several incorporated components. Therefore our objective is not limited to the estimation of the security perspective, but provides a reduction in alert redundancies and false positives,

in addition to the detection of multi-stage attacks and attack verification. The alerts are supplied by the IDS in real-time and then each alert is abstracted to an attack concept based on signature modelling in the knowledge-base. Pre- and post-condition mechanisms are applied to extract the relationships between events according to temporal information based on time context and spatial characteristics, e.g. the location of the detected activity and the vulnerability description.

Attack modelling: We have modelled attacks and attack capabilities on the basis of inheritance and abstraction principles. Specific attack descriptions provided by the IDS can reduce the detection domain and do not recognise the dependencies between alerts. Alerts are modelled to attack concept abstractions based on the status of the system being monitored. For example, consider two attack scenarios, m_1 and m_2, including the installation of some backdoor Trojans. Scenario m_1, in order to succeed, requires Trojan x to be installed, and scenario m_2 needs Trojan y to succeed. However, the IDS rules have a signature for Trojan x, whereas Trojan y is unknown. According to the hierarchical generalisation of attack capabilities, both scenarios will be categorized as Trojan activities due to the similarity of their effect on the targeted system. The Trojan installation action is just a single step among a sequence of stages in the performance of the attack.

We have developed a set of algorithms to employ for the proposed farmework:

Alert correlation algorithm

The inputs of the algorithm are the instances of detected attack capabilities in the form of encoded pre- and post-conditions. The intilization of these encoded conditions is performed based on an in-memory knowledge-base of the attack concepts. The correlation between the instances of elementary alerts is created according to partial matching of hierarchical multi-layer capability descriptions. The matching criteria are

based on the rules infered from the knowledge-base. The source addresses of certain attacks are considered whilst for others it is the destination address that is taken in account according to the signature direction. This technique is proposed in order to broaden the maximum detection coverage. This can result in a huge correlation of links; however, a supression mechanism to distinguish only those connections that are related is achieved using a vulnerabilty knowledge-base. In the intial stage, fine-grained correlations are identified and saved in a temporay container.

Alert verfication

To avoid degradation in the correlation process quality, which can be caused by having a false positive as an input, we have developed a filtering meachnism. A complementory algorithm was developed to examine the oppotunity of attack success according to vulnerability analyses and running services on the target system. This is an opposed technique to the generalisation mechanism of abstracting alerts in order to reduce false positives. Therefore the objective is to prioritize attacks based on their success, where failed attacks are assigned low priority. However, the failed attack is ignored unless it is considered as an attempt of real attack. The vulnerability knowledge is updated in a passive fashion to minimise communication load during the detection process.

Data reduction

The final result of the system is to produce a summarized attack graph in the form of generlised events. In practice, this graph should be concise and meaningful, disregarding insignificant details. The graph consists of nodes representing attack steps and edges to specify the logical connection between these temporally ordered steps. We have developed two algorithms for data reduction: one to minimise the number of involved redundancy nodes and the other to remove duplicate edges.

Graph reduction

Two algorithms have been developed to remove transitive edges from the resulting graph. We assume that the causal relationships propagate from root to leaf nodes. Online graph reduction is performed throughout the first stage of the correlation process. And offline algorithm is an optional, further reduction after the attack graph is built to ensure that we have the minimum available graph. In addition, this technique is used to restrain the relaxing mechanism in attack capabilities.

Alert aggregation

In contrast to graph reduction algorithms where the number of nodes is not affetcted, here nodes are aggregated. We aggregate two alerts according to their similarity, e.g. attack type and spatial and temporal characteristics. We also use a window time to determine the temporal proximity of alerts that can be aggregated.

Event generation algorithm

Simultaneously during the correlation process, *infall events* are generated if at least two correlated alerts are identified. Moreover, every detected *infall event* is examined if it corresponds to a previous one. However, the recognized event is not reported untill the resulting information is reduced and aggregated. The aggregated and verfied alerts are linked to compose a new event representing the detected multi-stage attack.

Near real-time detection

Real-time correlation systems have been investigated in several research efforts. Most of these methods tend to allocate memory space to store a bulk of states for a period of time which is naturally finite. This can be useful for a limited time but is vulnerable to detection avoidance. On the other hand, keeping a large amount of states in memory becomes problematic if the number of states increases. All previous works assume that

related activities fall in a short time period, e.g. a few hours in the best-case scenario. Therefore, we have developed a near real-time system to overcome the slow-and-low attack. This technique leverages the analysis domain to include even very old activities. However, to sustain higher performance, the alert data is maintained every time the analysis executed.

7.3.1 Avenues for future research

Along the development of our framework, we have identified certain directions for the improvement of the system's functioning.

1- Knowledge-based correlation approaches are precise and generate less false positives. They require the description of every possible capability and map IDS signatures to these capabilities. Probablistic approaches can uncover unidentified relationships but they may produce false relationships. An amalgamation of both techniques can be used in order to exploit their advantages. However, statistical analyses can be employed to compute the similarities between alerts, and the knowledge-base can be used to validate these decisions. This notion is borrowed from the amalgamation of anomaly-based IDSs with signature-based IDSs. Anomaly-based approaches have been investigated for several years but are still immature. Even though some vendors claim that they are employing these techniques in their products, they are considered black boxes. Therefore the use of an anomaly-based IDS as a standalone system is impractical in terms of the high number of false positives generated. We believe that such collaboration needs further investigation in order to facilitate the correlation process.

2- We have adopted a passive approach to vulnerability acquisition, which is useful in providing knowledge about protected networks. Instead, active analysis of the end-host response for the attacker can be considered to yield more precise results. This is

due to the information supplied by the vulnerability scanner being insufficient for the formulation of decisions regarding the success of attacks. The victim's point of view about the attack can support the analysis process by providing accurate details, e.g. whether the target port accepts the connection or not, which is not identified by the normal scanner.

3- As the correlation accuracy can be improved with the involvement of the maximum amount of available information, host-based IDS alerts may be used. In addition, Web application IDS tools can also utilized to obtain reliable and true observations. The incorporation of such systems requires a normalization stage to unify the supplied information.

4- The problem in terms of the discrimination between different attackers attacking the same target machine has been a debatable issue in the field of alert correlation. It is very difficult to decide whether or not a group of attackers are cooperating. Reliance on the source IP address is not feasible, as a single attacker can use several spoofed IP addresses. Therefore a behaviour analysis that is not based on traditional information is required. Analysis of only temporal characteristics is unrealistic because, as stated on many occasions throughout this thesis, a skilful attacker's activities can be conducted over long periods of time.

7.4 Implementation and evaluation of our framework

The proposed algorithms have been implemented in MARS tools to validate their practical application. The MARS server represents the core of the system that performs the functions of the collaborated components. Knowledge bases are stored in the memory and the MARS database is interacted with periodically to handle received

alerts. We have also developed extensible client tools for the administrator to obtain reports of multi-stage attacks in a visualised format.

We have evaluated the proposed system using a variety of datasets and by conducting real-life scenarios. We have also explained the evaluation metrics and applied these to obtain reliable results. Functional, accuracy and reduction evaluations have been implemented on all test categories. The results obtained have shown significant progress among all test parameters. For instance, the testing of the DARPA datasets yielded a 96.3% accuracy rate and a 99.1% reduction rate for INSIDE1.0. The alert verification mechanism has raised the overall accuracy among all conducted scenarios. Moreover, performance evaluation has also been elaborated using offline and online tests. The results have illustrated that MARS consumes less resources than Snort in both tests. We can conclude that the application of MARS does not affect the overall IDS performance.

Bibliography

[1]. D.S. Wall, "Cybercrime, media and insecurity: The shaping of public perceptions of cybercrime," *International Review of Law, Computers & Technology*, vol. 22, no. 1, 2008, pp. 45-63.

[2]. S.W. Lodin and C.L. Schuba, "Firewalls fend off invasions from the Net," *IEEE Spectrum*, vol. 35, no. 2, 1998, pp. 26-34.

[3]. J.C. Perez, "Gartner: Security concerns to stunt e-commerce growth," *IDG News Service*, 2005.

[4]. K.P. Yee, "Aligning security and usability," *Security & Privacy, IEEE*, vol. 2, no. 5, 2004, pp. 48-55.

[5]. R. Trost, *Practical intrusion analysis: prevention and detection for the twenty-first century*, Addison-Wesley Professional, 2009.

[6]. G. Vasiliadis, S. Antonatos, M. Polychronakis, E. Markatos and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," Springer, 2008, pp. 116-134.

[7]. P. Fogla, "Improving the efficiency and robustness of intrusion detection systems," PhD, Georgia Institute of Technology, Atlanta, GA, USA, 2007.

[8]. M. Akhlaq, F. Alserhani, A. Subhan, I.U. Awan, J. Mellor and P. Mirchandani, "High Speed NIDS using Dynamic Cluster and Comparator Logic," *Proc. the 2010 10th IEEE International Conference on Computer and Information Technology(CIT)*, IEEE Computer Society, 2010, pp. 575-581.

[9]. A.A. Ghorbani, *Network Intrusion Detection and Prevention: Concepts and Techniques*, Springer-Verlag New York Inc, 2009.

[10]. K. Julisch, "Using Root Cause Analysis to Handle Intrusion Detection Alarms," PhD, University of Dortmund, Germany, 2003.

[11]. T.H. Ptacek and T.N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," *Secure Networks, Inc., Jan*, 1998.

[12]. M. Handley, V. Paxson and C. Kreibich, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," *Proc. the 10th conference on USENIX Security Symposium*, USENIX Association, 2001, pp. 9.

[13]. C. Kruegel, F. Valeur, G. Vigna and R. Kemmerer, "Stateful intrusion detection for high-speed network's," *Proc. the IEEE symposium on Security and Privacy*, IEEE, 2005, pp. 285-293.

[14]. I. Charitakis, K. Anagnostakis and E. Markatos, "An active traffic splitter architecture for intrusion detection," *Proc. the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems(MASCOTS 2003). ,* 2003, pp. 238-241.

[15]. M. Aldwairi, T. Conte and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 1, 2005, pp. 99-107.

[16]. C. Clark, W. Lee, D. Schimmel, D. Contis, M. Koné and A. Thomas, "A hardware platform for network intrusion detection and prevention," *Network Processor Design: Issues and Practices*, vol. 3, 2004, pp. 99–118.

[17]. B.L. Hutchings, R. Franklin and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," *Proc. the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 111-120.

[18]. H. Youm and J. Lee, "Web 2.0 Security Technology Trends and Promoting Standardization," *Journal of Telecommunications Technology Association*, vol. 117, 2008, pp. 21-29.

[19]. G. Young and J. Pescatore, "Magic Quadrant for Network Intrusion Prevention System Appliances," *Gartner RAS Core Research Note G*, vol. 167309, 2009.

[20]. M. Roesch, "Snort–Lightweight Intrusion Detection for Networks," *Proc. the 13th USENIX conference on System administration*, USENIX Association, 1999, pp. 229-238.

[21]. V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23, 1999, pp. 2435-2463.

[22]. J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, 2000, pp. 262-294.

[23]. "Snort "; http://www.snort.org/.

[24]. J. Beale, A.R. Baker and J. Esler, *Snort IDS and IPS Toolkit*, Syngress Publishing, 2007.

[25]. G. Vigna, W. Robertson and D. Balzarotti, "Testing network-based intrusion detection signatures using mutant exploits," *Proc. the 11th ACM conference on*

Computer and communications security, ACM Press New York, NY, USA, 2004, pp. 21-30.

[26]. D. Maynor and K.K. Mookhey, *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*, Syngress Press, 2007.

[27]. L. Schaelicke, T. Slabach, B. Moore and C. Freeland, "Characterizing the Performance of Network Intrusion Detection Sensors," *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 2820, Springer Berlin / Heidelberg, 2003, pp. 155-172.

[28]. A. Valdes and K. Skinner, "Probabilistic Alert Correlation," *Proc. the 4th International Symposium on Recent Advances in Intrusion Detection* Springer-Verlag, 2001, pp. 54-68.

[29]. X. Qin, "A probabilistic-based framework for infosec alert correlation," PhD, Georgia Institute of Technology, 2005.

[30]. S. Lee, B. Chung, H. Kim, Y. Lee, C. Park and H. Yoon, "Real-time analysis of intrusion detection alerts via correlation," *Computers & Security*, vol. 25, no. 3, 2006, pp. 169-183.

[31]. H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts," *Proc. the 4th International Symposium on Recent Advances in Intrusion Detection* Springer-Verlag, 2001, pp. 85-103.

[32]. F. Cuppens, "Managing Alerts in a Multi-Intrusion Detection Environment," *Proc. Third International Workshop on Recent Advances in Intrusion Detection*, IEEE Computer Society, 2001, pp. 197-216.

[33]. F. Cuppens and R. Ortalo, "LAMBDA: A language to model a database for detection of attacks," *Proc. 17th Annual Computer Security Applications Conference* Springer, 2001, pp. 197-216.

[34]. P. Ning and D. Xu, "Toward Automated Intrusion Alert Analysis," *Network Security*, 2010, pp. 175-205.

[35]. P. Ning, Y. Cui, D.S. Reeves and D. Xu, "Techniques and tools for analyzing intrusion alerts," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 2, 2004, pp. 318.

[36]. K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," *Proc. the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2002, pp. 366-375.

[37]. A. Zhang, Z. Li, D. Li and L. Wang, "Discovering novel multistage attack patterns in alert streams," *Proc. International Conference on Networking, Architecture, and Storage-Cover*, IEEE, 2007, pp. 115-121.

[38]. B. Zhu and A.A. Ghorbani, "Alert correlation for extracting attack strategies," *International Journal of Network Security*, vol. 3, no. 2, 2006, pp. 244-258.

[39]. "Common Vulnerabilities and Exposures (CVE)," http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0188.

[40]. N. Paulauskas and J. Skudutis, "Investigation of the Intrusion Detection System "Snort" Performance," *Electronics and Electrical Engineering*, vol. 7, no. 87, 2008, pp. 15-18.

[41]. "NSS Labs," http://nsslabs.com/.

[42]. W. Lee, J.B.D. Cabrera, A. Thomas, N. Balwalli, S. Saluja and Y. Zhang, "Performance adaptation in real-time intrusion detection systems," *Proc. the 5th international conference on Recent advances in intrusion detection*, Springer-Verlag, 2002, pp. 252-273.

[43]. L. Schaelicke, K. Wheeler and C. Freeland, "SPANIDS: a scalable network intrusion detection loadbalancer," *Proc. the 2nd conference on Computing frontiers*, ACM, 2005, pp. 315-322.

[44]. M. Aldwairi, "Hardware Efficient Pattern Matching Algorithms and Architectures for Fast Intrusion Detection," PhD, North Carolina State University, 2006.

[45]. V. Paxson, K. Asanovic, S. Dharmapurikar, J. Lockwood, R. Pang, R. Sommer and N. Weaver, "Rethinking hardware support for network analysis and intrusion prevention," *Proc. the 1st USENIX Workshop on Hot Topics in Security* USENIX Association, 2006, pp. 11-11.

[46]. S. Axelsson, *Intrusion detection systems: A survey and taxonomy*, Technical Report, Chalmers University of Technology, Dept. of Computer Engineering, 2000.

[47]. J. Riordan, D. Zamboni and Y. Duponchel, "Billy goat, an accurate worm-detection system," *Research Report RZ3609, IBM Research GbmH,. Zurich Research Laboratory*, 2005.

[48]. F. Valeur, D. Mutz and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks," *Intrusion and Malware Detection and Vulnerability Assessment*, Lecture Notes in Computer Science 3548, Springer Berlin / Heidelberg, 2005, pp. 533-546.

[49]. F. CUPPENS and A. MIEGE, “Alert correlation in a cooperative intrusion detection framework,” *Proc. the 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 202-215.

[50]. A. Wespi and H. Debar, “Aggregation and Correlation of Intrusion-Detection Alerts,” *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 2212, Springer Berlin / Heidelberg, 2001, pp. 85-103.

[51]. F. Valeur, G. Vigna, C. Kruegel and R.A. Kemmerer, “A Comprehensive Approach to Intrusion Detection Alert Correlation,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, 2004, pp. 273--318.

[52]. A. Todd, R. Raines, R. Baldwin, B. Mullins and S. Rogers, “Alert Verification Evasion Through Server Response Forging,” *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 4637, Springer Berlin / Heidelberg, 2007, pp. 256-275.

[53]. G. Tedesco and U. Aickelin, “Real-Time Alert Correlation with Type Graphs,” *Information Systems Security*, Lecture Notes in Computer Science 5352, Springer Berlin / Heidelberg, 2008, pp. 173-187.

[54]. “DEFCON : Hacking conference,” http://www.defcon.org/.

[55]. “MIT Lincoln Laboratory ”; http://www.ll.mit.edu/.

[56]. A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur and J. Srivastava, “A comparative study of anomaly detection schemes in network intrusion detection,” *Proc. the Third SIAM International Conference on Data Mining* 2003, pp. 25–36.

[57]. V. Kumar, J. Srivastava and A. Lazarevic, “Intrusion Detection: A Survey,”

Managing Cyber Threats, Massive Computing 5, Springer US, 2005, pp. 19-78.

[58]. R. Bace and P. Mell, *Intrusion detection systems*, US Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2001.

[59]. I. Ristic, *Apache security*, O'Reilly Media, Inc., 2005.

[60]. J.P. Anderson, *Computer security threat monitoring and surveillance*, Technical Report, James P Anderson Co. FortWashington, Pennsylvania,USA, 1980.

[61]. D.E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, vol. SE-13, no. 2, 1987, pp. 222-232.

[62]. H. Debar, M. Dacier and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, 1999, pp. 805-822.

[63]. J.M. Estevez-Tapiador, P. Garcia-Teodoro and J.E. Diaz-Verdejo, "Anomaly detection methods in wired networks: a survey and taxonomy," *Computer communications*, vol. 27, no. 16, 2004, pp. 1569-1584.

[64]. P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, 2009, pp. 18-28.

[65]. S. Zanero, "Detecting 0-day attacks with learning intrusion detection system," *Blackhat Briefings, USA*, 2004.

[66]. S.E. Smaha, T.A.S. Inc and T.X. Austin, "Haystack: An intrusion detection system," *Proc. the IEEE fourth Aerospace Computer Security Applications Conference*, IEEE Computer Society Press, 1988, pp. 37-44.

[67]. N. Ye, S.M. Emran, Q. Chen and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Transactions on Computers & Security*, vol. 51, no. 7, 2002, pp. 810-820.

[68]. D.E. Denning, D. Edwards, R. Jagannathan, T. Lunt and P. Neumann, "A Prototype IDES—A Real-Time Intrusion Detection Expert System," *Computer Science Laboratory, SRI International*, 1987.

[69]. T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P.G. Neumann and C. Jalali, "IDES: a progress report [Intrusion-Detection Expert System]," *Proc. the Sixth Annual Computer Security Applications Conference*, IEEE Computer Society Press, 1990, pp. 273-285.

[70]. H.S. Vaccaro and G.E. Liepins, "Detection of anomalous computer session activity," *Proc. IEEE Symposium on Security and Privacy*, 1999, pp. 280-289.

[71]. S. Staniford, J.A. Hoagland and J.M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1, 2002, pp. 105-136.

[72]. M. Bishop, *Introduction to computer security*, Addison-Wesley Professional, 2004.

[73]. B.V. Nguyen, *An application of support vector machines to anomaly detection*, Technical Report CS681, Ohio University, Athens, OH, USA, 2002.

[74]. S. Forrest, S.A. Hofmeyr and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, 1998, pp. 151-180.

[75]. D. Mutz, F. Valeur, G. Vigna and C. Kruegel, "Anomalous system call detection,"

ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 1, 2006, pp. 61-93.

[76]. C. Kruegel, D. Mutz, W. Robertson and F. Valeur, "Bayesian event classification for intrusion detection," *Proc. the 19th Computer Security Applications Conference*, IEEE Computer Society, 2003, pp. 14-23.

[77]. A. Valdes and K. Skinner, "Adaptive, Model-Based Monitoring for Cyber Attack Detection," *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 1907, Springer Berlin / Heidelberg, 2000, pp. 80-93.

[78]. D. Barbara, N. Wu and S. Jajodia, "Detecting novel network intrusions using bayes estimators," *Proc. the First SIAM Conference on Data Mining*, 2001.

[79]. M.L. Shyu, S.C. Chen, K. Sarinnapakorn, L.W. Chang, E. Miami Univ Coral Gables Fl Dept Of and E. Computer, *A novel anomaly detection scheme based on principal component classifier*, Defense Technical Information Center, 2003.

[80]. W. Wang, X. Guan and X. Zhang, "Processing of massive audit data streams for real-time anomaly intrusion detection," *Computer Communications*, vol. 31, no. 1, 2008, pp. 58-72.

[81]. K. Dong Seong, N. Ha-Nam, T. Thandar and P. Jong Sou, "An Optimized Intrusion Detection System Using PCA and BNN," *Proc. the 6th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT 2005)* 2005, pp. 356-359.

[82]. N. Ye, X. Li, Q. Chen, S.M. Emran and M. Xu, "Probabilistic techniques for intrusion detection based on computer audit data," *IEEE Transactions on Systems,*

Man and Cybernetics, Part A, vol. 31, no. 4, 2001, pp. 266-274.

[83]. D.Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, no. 1, 2003, pp. 229-244.

[84]. S.B. Cho and H.J. Park, "Efficient anomaly detection by modeling privilege flows using hidden Markov model," *Computers & Security*, vol. 22, no. 1, 2003, pp. 45-55.

[85]. M.V. Mahoney, "A machine learning approach to detecting attacks by identifying anomalies in network traffic," PhD, Florida Institute of Technology, 2003.

[86]. W. Lee and S.J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security*, vol. 3, no. 4, 2000, pp. 227-261.

[87]. J.E. Dickerson and J.A. Dickerson, "Fuzzy network profiling for intrusion detection," *Proc. the 19th International Conference of the North American on Fuzzy Information Processing Society*, 2000, pp. 301-306.

[88]. S.M. Bridges and R.B. Vaughn, "Fuzzy data mining and genetic algorithms applied to intrusion detection," *Proc. the National Information Systems Security Conference*, 2000, pp. 13-31.

[89]. C. Gates, "Co-ordinated port scans: a model, a detector and an evaluation methodology," PhD, Dalhousie University, Halifax, Canada, 2006.

[90]. S. Ramaswamy, R. Rastogi and K. Shim, "Efficient algorithms for mining outliers from large data sets," *ACM SIGMOD Record*, vol. 29, no. 2, 2000, pp. 427-438.

[91]. E.M. Knorr, "Outliers and data mining: Finding exceptions in data," PhD, The

University of British Columbia (Canada), 2002.

[92]. L. Ertoz, E. Eilertson, A. Lazarevic, P.N. Tan, V. Kumar, J. Srivastava and P. Dokas, "Minds-minnesota intrusion detection system," *Proc. Next Generation Data Mining Challenges and Directions*, MIT Press, 2004.

[93]. D. Barbará, J. Couto, S. Jajodia and N. Wu, "ADAM: A testbed for exploring the use of data mining in intrusion detection," *ACM SIGMOD Record*, vol. 30, no. 4, 2001, pp. 15-24.

[94]. D.F. Gong, "White Paper: Deciphering Detection Techniques: Part II Anomaly-based Intrusion Detection," *Network Associates (McAfee Security)*, 2003.

[95]. F. Anjum, D. Subhadrabandhu and S. Sarkar, "Signature based intrusion detection for wireless ad-hoc networks: a comparative study of various routing protocols," *Proc. the IEEE 58th on Vehicular Technology Conference(VTC 2003)*, 2003, pp. 2152-2156 Vol.2153.

[96]. G.A. Stephen, *String searching algorithms*, World Scientific Pub Co Inc, 1994.

[97]. R.T. Liu, N.F. Huang, C.H. Chen and C.N. Kao, "A fast string-matching algorithm for network processor-based intrusion detection system," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 3, 2004, pp. 614-633.

[98]. J. Beale, A.R. Baker, J. Esler, T. Kohlenberg and S. Northcutt, *Snort: IDS and IPS toolkit*, Syngress Media Inc, 2007.

[99]. F. Yu, "High speed deep packet inspection with hardware support," PhD, University of California, 2006.

[100]. R.S. Boyer and J.S. Moore, “A fast string searching algorithm,” *Communications of the ACM*, vol. 20, no. 10, 1977, pp. 762-772.

[101]. A.V. Aho and M.J. Corasick, “Efficient string matching: an aid to bibliographic search,” *Communications of the ACM*, vol. 18, no. 6, 1975, pp. 333-340.

[102]. A. Barkalov and L. Titarenko, *Logic synthesis for compositional microprogram control units*, Springer Verlag, 2008.

[103]. E.W. Spitznagel, “High Performance Packet Classification,” PhD, Washington University, 2005.

[104]. T. Abbes, A. Bouhoula and M. Rusinowitch, “On the fly pattern matching for intrusion detection with Snort,” *Annals of telecommunications*, vol. 59, no. 9, 2004, pp. 1045-1071.

[105]. S. Rubin, S. Jha and B.P. Miller, “Protomatching network traffic for high throughputnetwork intrusion detection,” *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 47-58.

[106]. D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws* John Wiley & Sons 2007.

[107]. E. Chickowski, “Don’t Fear the Unknown: Behavior Analysis Intrusion Prevention Defends Against Zero-Day Attacks.,” 2006; http://www.processor.com/editorial/article.asp?article=articles%2Fp2817%2F32p17%2F32p17.asp.

[108]. “Request for Comments (RFC),” http://www.ietf.org/rfc.html.

[109]. J.D. Case, M. Fedor, M.L. Schoffstall and J. Davin, "RFC1157: Simple network management protocol (SNMP)," *Internet RFCs*, 1990.

[110]. S. Wu, U. Manber and E. Myers, "A subquadratic algorithm for approximate regular expression matching," *Journal of algorithms*, vol. 19, no. 3, 1995, pp. 346-360.

[111]. "Snort-AI (Snort with Artificial Intellegence) "; http://snort-ai.sourceforge.net/.

[112]. "The Lawrence Berkeley National Laboratory," http://www.lbl.gov/.

[113]. M. Sipser, *Introduction to the Theory of Computation*, International Thomson Publishing, 1996.

[114]. I.Tripwire, "Tripwire changing monitoring and reporting solutions," http://www.tripwire.com/it-compliance-products/te/ost/.

[115]. C.A.D.B. Cid, "Ossec, open source host-based intrusion detection system," http://www.ossec.net/.

[116]. H. Debar, D. Curry and B. Feinstein, "RFC4765: The Intrusion Detection Message Exchange Format (IDMEF)," IETF, 2006.

[117]. S.T. Eckmann, G. Vigna and R.A. Kemmerer, "STATL: An attack language for state-based intrusion detection," *Journal of Computer Security*, vol. 10, no. 1, 2002, pp. 71-103.

[118]. E. Michel and M. Ludovic, "ADeLe: An attack description language for knowledge-based intrusion detection," *Proc. the 16th international conference on Information security: Trusted information: the new decade challenge* Kluwer, 2001,

pp. 353-368.

[119]. O. Dain and R.K. Cunningham, "Fusing a heterogeneous alert stream into scenarios," *Proc. the 2001 ACM workshop on Data Mining for Security Applications*, Citeseer, 2001, pp. 1–13.

[120]. B. Morin, L. Mé, H. Debar and M. Ducassé, "M2D2: A formal data model for IDS alert correlation," *Proc. the 5th international conference on Recent advances in intrusion detection (RAID'02)*, Springer-Verlag, 2002, pp. 115-137.

[121]. S.J. Templeton and K. Levitt, "A requires/provides model for computer attacks," *Proc. the 2000 workshop on New security paradigms*, ACM, 2000.

[122]. Z. Li, J. Lei, L. Wang and D. Li, "A data mining approach to generating network attack graph for intrusion prediction," *Proc. Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)* IEEE, 2007, pp. 307-311.

[123]. J. Ma, Z. Li and W. Li, "Real-Time Alert Stream Clustering and Correlation for Discovering Attack Strategies," *Proc. the 5th International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE, 2008, pp. 379-384.

[124]. L. Zhitang, Z. Aifang, L. Jie and W. Li, "Real-Time Correlation of Network Security Alerts," *Proc. the IEEE International Conference on e-Business Engineering(ICEBE 2007).* 2007, pp. 73-80.

[125]. R. Agrawal and R. Srikant, "Mining sequential patterns," *Proc. the Eleventh International Conference on Data Engineering, 1995.*, 1995, pp. 3-14.

[126]. N. Desai, "IDS correlation of VA data and IDS alerts," *Security Focus* June 2003; www.securityfocus.com/infocus/1708.

[127]. R. Gula, *Correlating ids alerts with vulnerability information*, Technical Report, Tenable Network Security, 2002.

[128]. "Nessus: Security Scanner," http://www.nessus.org.

[129]. P. Porras, M. Fong and A. Valdes, "A Mission-Impact-Based Approach to INFOSEC Alarm Correlation," *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 2516, Springer Berlin / Heidelberg, 2002, pp. 95-114.

[130]. G.F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure, USA, 2009.

[131]. L. Wang, A. Liu and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Computer communications*, vol. 29, no. 15, 2006, pp. 2917-2933.

[132]. P. Mell, V. Hu, R. Lippmann, J. Haines and M. Zissman, "An overview of issues in testing intrusion detection systems," *NIST IR*, vol. 7007, 2003.

[133]. "D-ITG V 2.6 "; http://www.grid.unina.it/Traffic/index.php.

[134]. "LAN Traffic V 2 "; http://www.topshareware.com

[135]. "Karalon's Traffic IQ Pro "; http://www.karalon.com/trafficiqpro.htm.

[136]."SmartBits," http://www.spirent.com.

[137]. J. Sommers, H. Kim and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, 2004, pp. 392-392.

[138]. L.M. Rossey, R.K. Cunningham, D.J. Fried, J.C. Rabek, R.P. Lippmann, J.W.

Haines and M.A. Zissman, “LARIAT: Lincoln adaptable real-time information assurance testbed,” *Proc. the IEEE Aerospace Conference* 2002, pp. 2671-2676, 2678-2682.

[139]. “CANVAS ”; http://www.immunitysec.com/products-canvas.shtml.

[140]. F. Massicotte, F. Gagnon, Y. Labiche, L. Briand and M. Couture, “Automatic evaluation of intrusion detection systems,” *Proc. the 22nd Annual Computer Security Applications Conference(ACSAC'06).* 2006, pp. 361-370.

[141]. L. Juan, C. Kreibich, C.H. Lin and V. Paxson, “A Tool for Offline and Live Testing of Evasion Resilience in Network Intrusion Detection Systems,” Springer, 2008, pp. 267-278.

[142]. “Infrastructure Security Report- ARBOR Networks,” 2010; http://www.arbornetworks.com/report.

[143]. E. Verplanke, “Understand packet-processing performance when employing multicore processors,” *Embedded Systems Design*, vol. 20, no. 4, 2007, pp. 36.

[144]. T. Vermeiren, E. Borghs and B. Haaodorens, “Evaluation of software techniques for parallel packet processing on multi-core processors,” *Proc. the First IEEE Consumer Communications and Networking Conference(CCNC 2004).* , 2004, pp. 645-647.

[145]. F. Schneider, J. Wallerich and A. Feldmann, “Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware,” *Lecture Notes in Computer Science*, vol. 4427, 2007, pp. 207.

[146]. L. Foschini, A. Thapliyal, L. Cavallaro, C. Kruegel and G. Vigna, “A Parallel

Architecture for Stateful, High-Speed Intrusion Detection," *Information Systems Security*, 2008, pp. 203-220.

[147]. "ProCurve Series 2900 Switch," http://www.hp.com

[148]. "NetCPS. ," http:// www.netchain.com/NetCPS/.

[149]. "Tfgen.," http:// www.st.rim.or.jp/~yumo/pub/tfgen.

[150]. "Http Traffic Generator.," http://www.nsauditor.com/.

[151]. "Hping V 2," http://www.hping.org/download.html.

[152]. "Bandwidth Monitor "; http://sourceforge.net/projects

[153]. "Nload "; http://www.sourceforge.net/projects/nload/.

[154]. A. Singh, "An introduction to virtualization.," 2004; http://www.kernelthread.com/publications/virtualization.

[155]. "Business value of virtualization: Realizing the benefits of integrated solutions," 2009;http://h18000.www1.hp.com/products/servers/management/vse/Biz_Virtualization_WhitePaper.pdf.

[156]. J. Xu, M. Zhao, J. Fortes, R. Carpenter and M. Yousif, "On the use of fuzzy modeling in virtualized data center management," *Proc. the 4th International Conference on Automotatic Computing (ICAC2007)*, IEEE, 2007, pp. 25.

[157]. "SATA Technology," http://www.serialata.org/.

[158]. "Disk Queue Length Counter," http://www.windowsnetworking.com/articlestutorials/Windows-Server-2003-

PerfTuning.html.

[159]. M. Akhlaq, F. Alserhani, I.U. Awan, J. Mellor, A.J. Cullen and P. Mirchandani, "Virtualization Efficacy for Network Intrusion Detection Systems in High Speed Environment," *Information Security and Digital Forensics*, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering 41, Springer Berlin Heidelberg, pp. 26-41.

[160]. J. Zhou, M. Heckman, B. Reynolds, A. Carlson and M. Bishop, "Modeling network intrusion detection alerts for correlation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 1, 2007, pp. 4-es.

[161]. J.W. Haines, R.P. Lippmann, D.J. Fried, E. Tran, S. Boswell and M.A. Zissman, *DARPA intrusion detection system evaluation: Design and procedures*, Technical Report, Lincoln Laboratory, Massachusetts Institute of Technology, 2000.

[162]. "Security Focus - BugTraq." http://www.securityfocus.com

[163]. X. Qin and W. Lee, "Attack plan recognition and prediction using causal networks," *Proc. the 20th Annual Computer Security Applications Conference (ACSAC'04)*, IEEE, 2005, pp. 370-379.

[164]. H.A. Kautz, "A formal theory of plan recognition and its implementation," *Reasoning about plans*, Morgan Kaufmann Publishers Inc., 1991, pp. 69-124.

[165]. "Developer Express," http://www.devexpress.com/.

[166]. J.W.S. Liu, *Real-time systems*, Prentice Hall, 2000.

[167]. S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection,"

ACM Transactions on Information and System Security (TISSEC), vol. 3, no. 3, 2000, pp. 186-205.

[168]. R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod and R.K. Cunningham, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," *Proc. DARPA Information Survivability Conference and Exposition(DISCEX'00)*, IEEE, 2002, pp. 12-26.

[169]. K.H. Zou, A.J. O'Malley and L. Mauri, "Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models," *Circulation*, vol. 115, no. 5, 2007, pp. 654.

[170]. D. Alessandri, "Attack-Class-Based Analysis of Intrusion Detection Systems," PhD, University of Newcastle, 2004.

[171]. R. Lippmann, E. Kirda, A. Trachtenberg, H. Dreger, A. Feldmann, V. Paxson and R. Sommer, "Predicting the Resource Consumption of Network Intrusion Detection Systems," *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 5230, Springer Berlin / Heidelberg, 2008, pp. 135-154.

[172]. E.M. Voorhees, D.K. Harman, N.I.o. Standards and Technology, *TREC: Experiment and evaluation in information retrieval*, MIT press Boston, 2005.

[173]. "iCAST/Acer eDC 2007 Intrusion Detection Alert Data Description," http://www.chmao.idv.tw/project/acer07.html.

[174]. "Colasoft Packet Player," http://www.colasoft.com/packet_player/.

[175]. F. Alserhani, M. Akhlaq, I. Awan, A. Cullen, J. Mellor and P. Mirchandani,

"Multi-Tier Evaluation of Network Intrusion Detection Systems," *Journal for Information Assurance and Security (JIAS)*, vol. 5, 2010, pp. 301 - 310.

[176]. "Basic Analysis and Security Engine," http://base.secureideas.net/.

[177]. K. Baylor and C. Brown, "Killing Botnets: A view from the trenches," October,2006; ttp://www.mcafee.com/us/local_content/white_papers/wp_botnet.pdf.

[178]. E. Stinson and J.C. Mitchell, "Towards systematic evaluation of the evadability of bot/botnet detection methods," *Proc. the 2nd Conference on USENIX Workshop on offensive Technologies*, USENIX Association, 2008, pp. 1-9.

[179]. H. Choi, H. Lee and H. Kim, "BotGAD: detecting botnets by capturing group activities in network traffic," *Proc. the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE(COMSWARE '09).* , ACM, 2009.

[180]. "Zeus Tracker," https://zeustracker.abuse.ch.

[181]. "Bleeding edge threats," http://www.bleedingthreats.net/.

[182]. "Emerging Threats," www.emergingthreats.net/.

[183]. "Open Web Application Security Project: OWASP Top Ten - Injection Flaws.," 2010; http://www.owasp.org/index.php/Top_10_2007-njection_Flaws.

Appendix I Snort signatures

An example of a Snort signature description:

```
Rule:
--
Sid:
610
--
Summary:
This event is generated when an attempt to login as the
superuser is attempted using rsh.
--
Impact:
Serious. If successful the attacker may have gained superuser
access to the host.
--
Detailed Information:
This rule generates an event when a connection is made using
"rsh" with the username "root". Such activity is indicative of
attempts to abuse insecure machines with a known default
configuration.

Some UNIX systems use the "rsh" daemon which permits remote
"root" logins. This may allow an attacker to connect to the
machine and establish an interactive session.
--
Attack Scenarios:
An attacker finds a machine with the "rsh" service running and
connects to it, then proceeds to guess the "root" password
--
Ease of Attack:
Simple, no exploit software required
--
False Positives:
A system administrator may be logging in to a host using the
username "root"
--
False Negatives:
If a local username is not the same as the remote one ("root"),
the rule will not generate an event.
--
Corrective Action:
Investigate logs on the target host for further details and more
signs of suspicious activity

Use ssh for remote access instead of rsh.

Deny remote root logins to the host, use a normal user and
"sudo" or give the user the ability to "su" to root where
appropriate.
```

Appendix II MARS GUI

1. MARS server interface

Figure.1 shows the MARS server interface with the following information:

Database IP : the IP address of the database server.

Server IP : the IP address and the operating port of the MARS server.

Client connections: the number of the connected MARS clients.

Analysis count: how many times the MARS server has analysed the database.

Last Analysis time: the time of the last analysis connection.

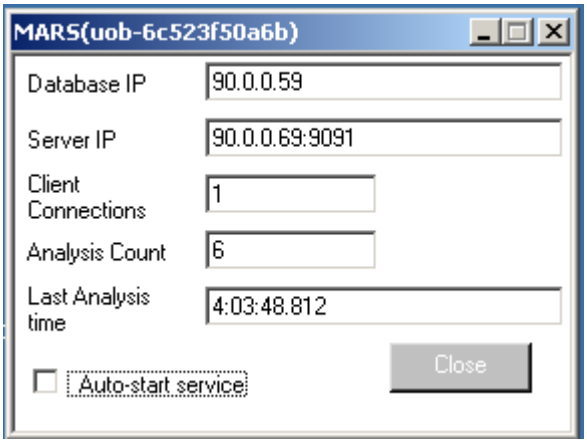

Figure.1 MARS server interface

2. MARS client interface

The Figures 2-4 show the MARS client interface displayed to the administrator. Evolving events are displayed as a list and old treated ones can be retrieved using events query. The available information through this form includes:

Event ID : a unique number to identify all detected events.

Event Title: to describe the attacking activities based on the attack category.

Priority: indicates the severity level of an event (High, Medium ,and Low).

Start-time and End-time: to show the start and the end time of the event, these times are not fixed as can be changed based the results of the event detection analysis.

Alert count: denotes the number of alerts involved in the detected event.

Steps: to identify the number of the attack stages.

Elapsed time: denotes the difference between the start and the end time of an event.

Confirmation: is used by the administrator if he thinks that an event is identified. If the detected event is not clear it is left until further information is received.

Dealing : this facility is used if an event needs more vulnerability investigation and risk analysis.

Close : when an event is identified, confirmed, and treated based on the organisation policy, it is closed to minimise the system process.

Each events listed on the top of the main form has its related alerts information listed on the bottom as shown in Figure.2. These alerts are aggregated and the number of involving alert instances is displayed.

The attack graph of each event can be displayed using the menu list as shown in Figure.4. The administrator can navigate each node to show its details. The detected attack steps (graph nodes) are ordered temporally from left to right and the edges show the causal relationships.

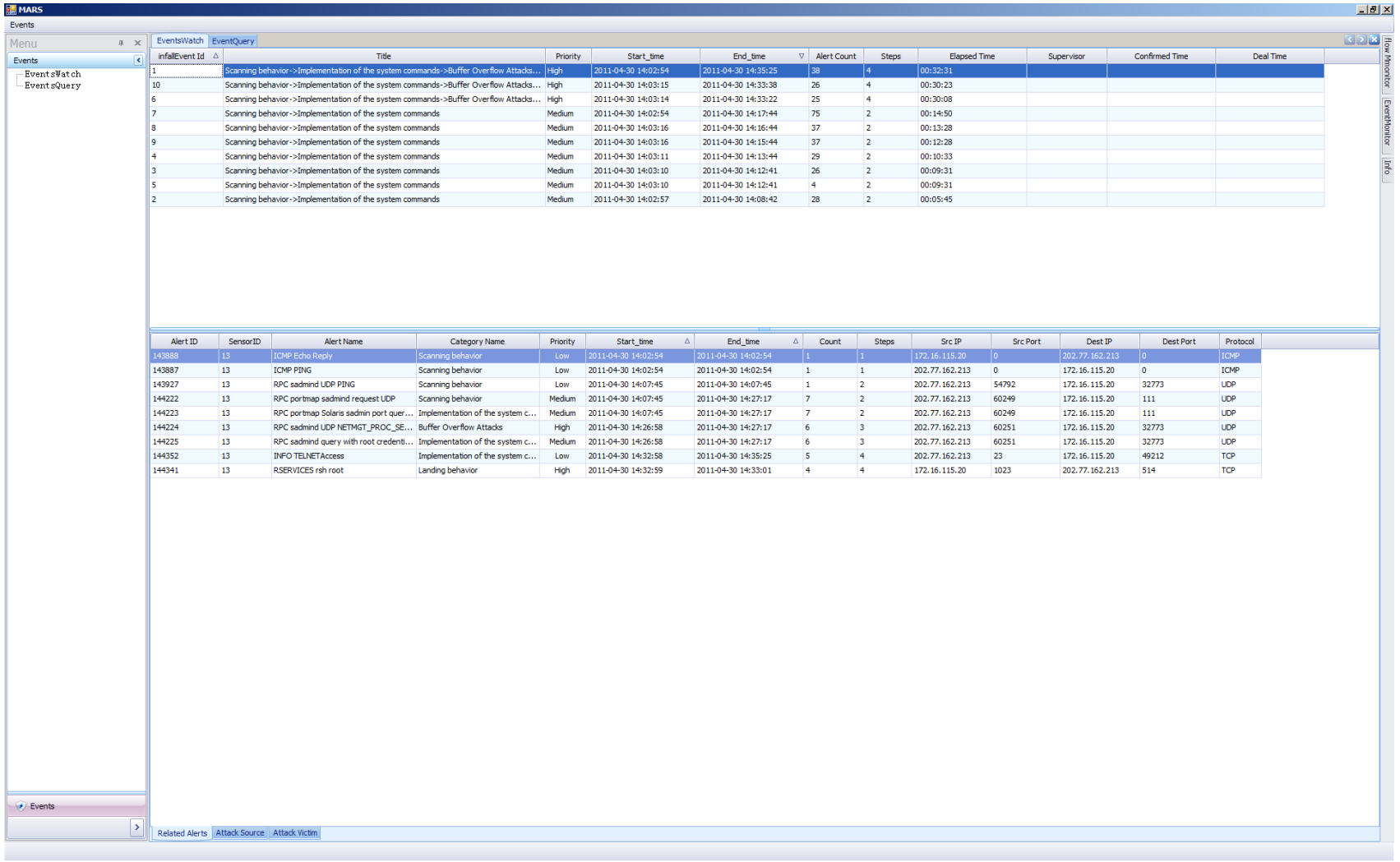

Figure.2 MARS client interface -1

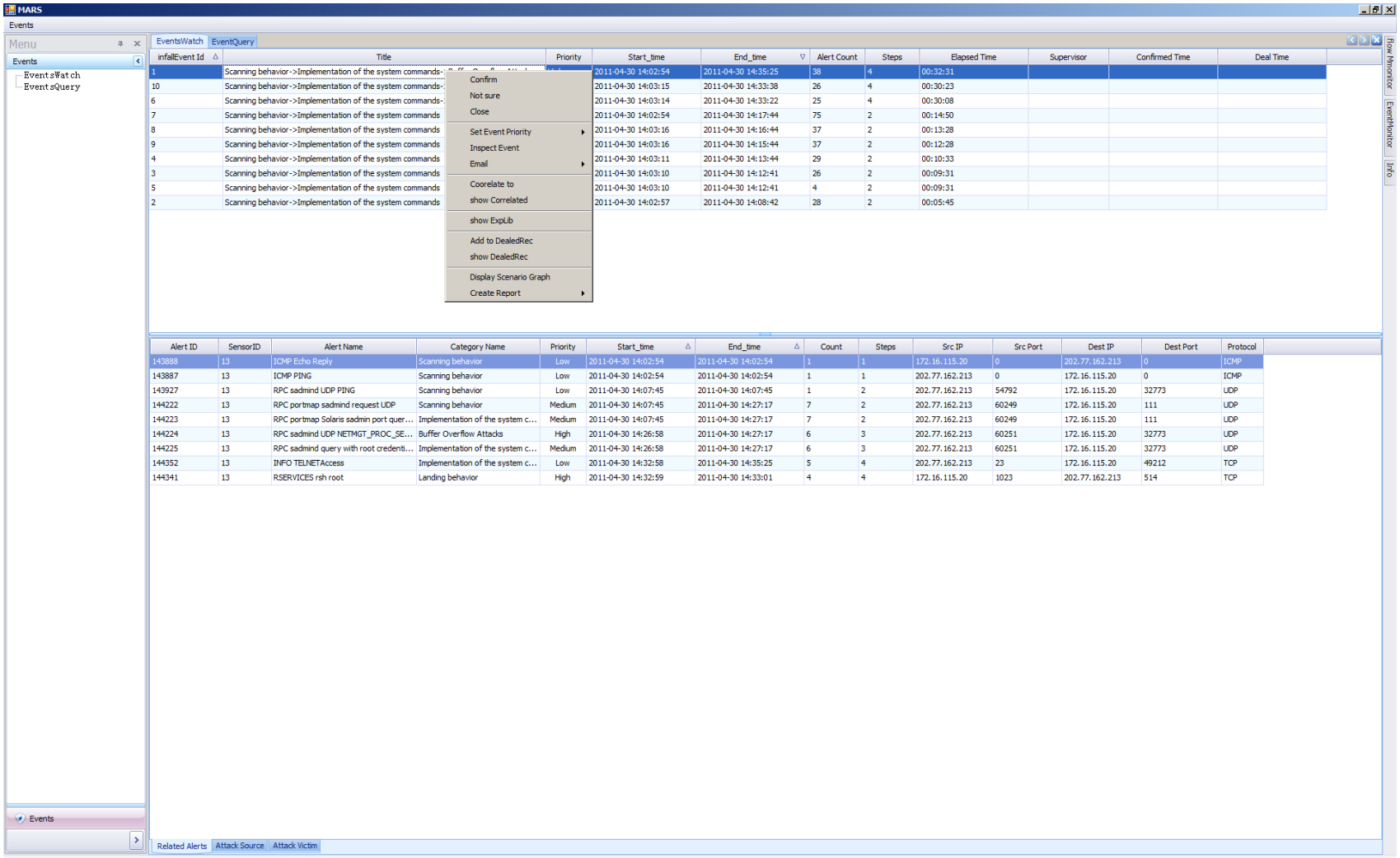

Figure.3 MARS client interface -2

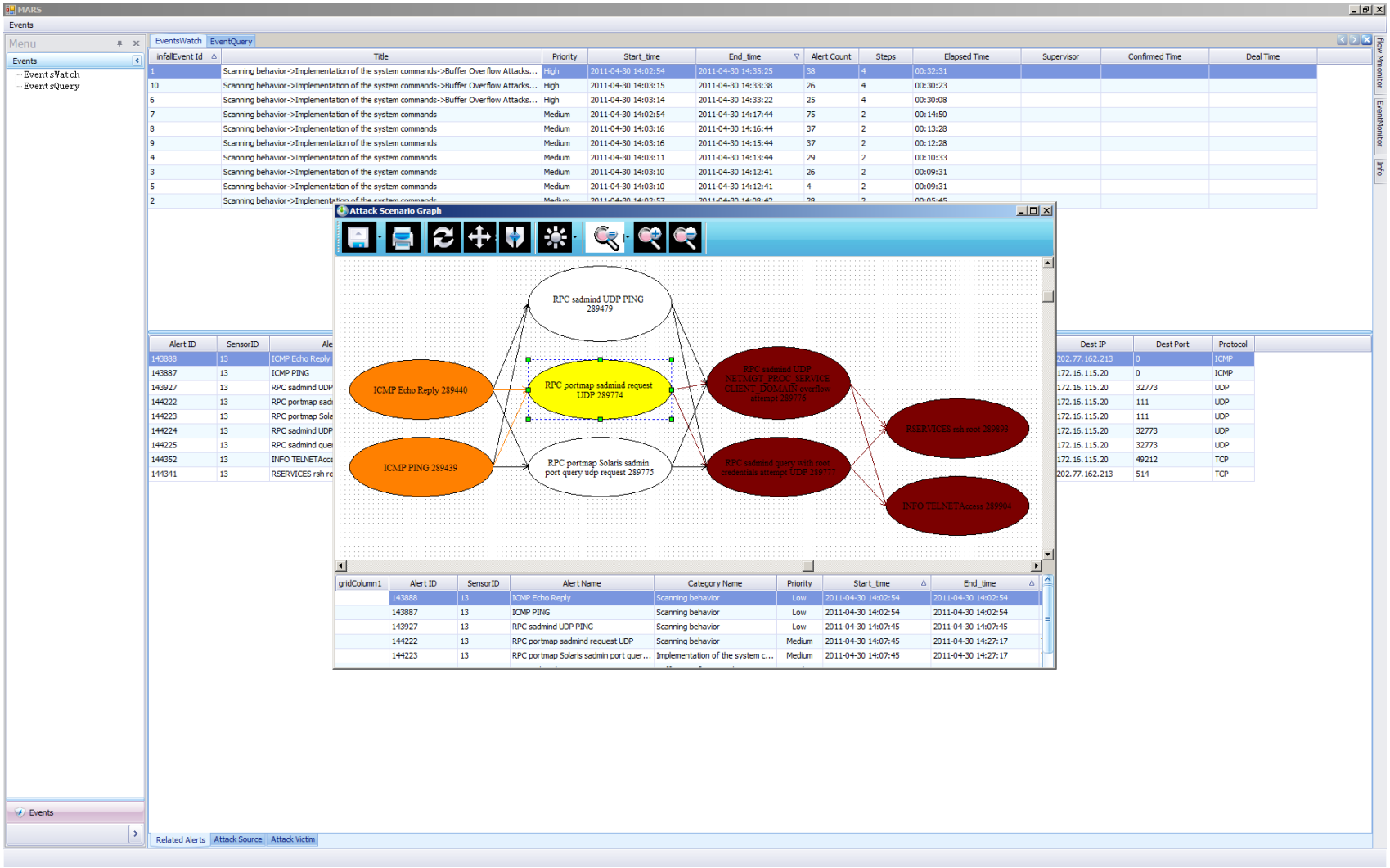

Figure.4 MARS client interface -3

Acronyms

BPF : Berkeley Packet Filter

CPU : Central Processing Unit

DARPA : Defence Advance Research Project Agency

DNS : Domain Name Server

DDoS : Distributed Denial of Service Attack

DoS: Denial of Service Attack

DMZ : Demilitarized Zone

FPGA : Field Programmable Graphical Array

FTP : File Transfer Protocol

GHz : Giga Hertz

GB : Giga Bytes

GUI : Graphical User Interface

Gbps : Giga bits per second

HIDS : Host based Intrusion detection System

HTTP : Hypertext Transfer Protocol

ID : Identification

IDS : Intrusion Detection Systems

IP : Internet Protocol

IP Address : Internet Protocol Address

IT : Information Technology

IDES : Intrusion Detection Expert System

IPS : Intrusion Prevention Systems

IRI : Informatics Research Institute

ICMP : Internet Control Message Protocol

I/O : Input/Output

IRC : Internet Relay Chat

LAN : Local Area Network

MARS: Multi-stage Attack Recognition System

Mbps : Mega bits per second

MB : Mega Bytes

NIDS : Network Intrusion Detection Systems

NSRG: Network Security Research Group

NIC : Network Interface Card

NAPI : New Application Program Interface

OS: Operating System

PCI : Peripheral Component Interconnect

PCIe : Peripheral Component Interconnect Express

RFC: Request for Comments

RPC : Remote Procedure Call

RAM : Random Access Memory

SNMP : Simple Network Management Protocol

SMTP : Simple Mail transfer Protocol

TCP: Transmission Control Protocol

UDP : User Datagram protocol

VLAN : Virtual Local Area Network