UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Facoltà di Ingegneria

Corso di Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XX Ciclo
Dipartimento di Informatica e Sistemistica

MULTIPLE CLASSIFIER SYSTEMS FOR NETWORK SECURITY
from data collection to attack detection

CLAUDIO MAZZARIELLO

Ph.D. Thesis

TUTOR
Prof. Carlo Sansone

COTUTOR
Prof. Simon Pietro Romano

COORDINATOR
Prof. Luigi Pietro Cordella

November 2007

Lo duca e io per quel cammino ascoso
intrammo a ritornar nel chiaro mondo;
e sanza cura aver d'alcun riposo,
salimmo sù, el primo e io secondo,
tanto ch'i' vidi de le cose belle
che porta 'l ciel, per un pertugio tondo.
E quindi uscimmo a riveder le stelle.

The Guide and I into that hidden road
Now entered, to return to the bright world;
And without care of having any rest
We mounted up, he first and I the second,
Till I beheld through a round aperture
Some of the beauteous things that Heaven doth bear;

Thence we came forth to rebehold the stars.

Dante Alighieri - Divina Commedia
Inferno, Canto XXXIV

Abstract

Since the Internet started developing, hosts and provided services have always been targeted with attacks trying to disrupt them. Trends show that, throughout the years, the number of hosts, as well as the degree of dependency of the whole society on the services provided through the Internet, increased dramatically, whereas the skills and knowledge required to interfere with normal network operation, and eventually to abruptly interrupt it, decreased accordingly.

This considerations urge the requirement for effective tools, aimed at granting security to Internet users. The need for systems capable of detecting attacks, and reacting in order to prevent them from occurring again, is nowadays undeniable.

In this thesis we propose methods based on multiple classifier systems for intrusion detection. We use such systems for automated data collection, also taking privacy issues into account. Some approaches to traffic classification are presented too, together with a proposal for the practical deployment of multiple classifiers in a real network environment.

Acknowledgments

This thesis marks the end of a long and tortuous process. It is the transition point between my life as a “student”, and the rest of it as a “professional”. At this point, I can see both the sacrifices I made, and the expectations for my future. Since I didn’t walk alone along this difficult path, I’d like to thank the people who have always been by my side.

I didn’t cite Dante Alighieri’s *Inferno* from *La Divina Commedia* by chance. During the last three years I’ve had my guides too. Like Virgil, they’ve been wisely advising me. They’ve taught me how to find new questions, and how to look for the answers. That’s why I’d like to thank my advisors, prof. Carlo Sansone and prof. Simon Pietro Romano, for all I’ve learned from them. Together with them, I’d also like to thank prof. Giorgio Ventre, for believing in me, and for all the lessons about how research can be both challenging and fun at the same time; that’s probably the greatest finding of all.

Sincere thanks go to prof. Wenke Lee, from the Georgia Institute of Technology. Together with my advisors, he allowed me to spend one year within his research group at the Georgia Tech Information Security Center of the College of Computing, in Atlanta, GA. This experience, and some of the people I met in Atlanta, gave me the opportunity to learn many new things, both about reseach and about myself as a man.

How could I forget both my colleagues, and fellow Ph. D. students, at the University of Napoli. They prove every day, how precious friendship is, especially at work, and how a laugh or a joke every now and then can help in solving the *serious problems facing mankind* [sic.], that researchers in computer science have to deal with, every day…I’d also especially like to thank one of them, Francesco Oliviero, with whom I shared these years of hard work. We’ve been arguing, yelling at each other sometimes, but in the end, we’ve always been working very well together.

I won’t forget my friends too. Despite my passion, my dedication, and my sense of commitment, they’ve kept suggesting me to find a serious job, instead of insisting on studying! We’ve known each other for years, and they know me better than many other people. I'm sure they will be as proud and extremely happy as I am for this.

Really special thanks go to my family, for supporting and "tolerating" me through the years. They never denied me any opportunity, and so far they've always encouraged me in pursuing my passions, my interests and my dreams. I've always been able to feel their love, their understanding, and their pride for my accomplishments. My gratitude for my father, my mother and my sister can't actually be expressed with words. Anything good I accomplish as a man, is also due to the way they raised me up, and for this I'm immensely grateful.

And last but not least, I'd like to thank my wife Caterina. We met when I was a student, and last September we got married. She's seen me happy, nervous, excited, frustrated. She always showed me unconditioned love and understanding, and especially during these last days, when I was really, REALLY stressed out, she patiently tolerated all my good and (sometimes extremely) bad moods. As I can imagine how it's been like, this gives me a stronger confirmation: if she's been able to live through this, it proves me how much she must love me. For all these reasons, and for the love I feel, this thesis is dedicated to her.

Contents

List of Figures

List of Tables

Chapter 1

Network Security: Threats and Tools

In recent years, studies on attack trends [97, 17] have shown the main evolution directions for attack strategies. The degree of automation and the speed of attack tools increases. Automated attacks typically scan for potential victims, looking for vulnerable systems to compromise. Previously, vulnerabilities were exploited after a widespread scan was complete. Now, attack tools exploit vulnerabilities as a part of the scanning activity, which increases the speed of propagation. Attacks automatically propagate, whereas before 2000 attack tools required a person to initiate additional attack cycles. Tools like Code Red and Nimda self-propagate to a point of global saturation in less than 18 hours. All the attack strategy is much more coordinated than it used to be in the past, giving rise to more dangerous consequences of successful attacks. An increase in sophistication of attack tools has been noticed. Modern attacks often include countermeasures against forensics, and also have the ability to perform mutation that make them virtually unrecognizable. While in the past scans for vulnerabilities where easily discovered by means of statistical rate-based analysis, nowadays scanning strategies have become more and more sophisticated, allowing for a faster discovery of vulnerabilities and a stealthier scanning. Also, attackers have changed their motivations. While in the past they were usually very skilled professionals, nowadays, with a wider distribution of computers, a lower expertise is required to perform very dangerous attacks (figure 1.1), thus virtually increasing the risk of being an attack victim. This is probably due to the fact that, though computers and the ability to interconnect them is today widespread, the awareness about the threats related to poor security policies are not well known. Many home or office computer users, for example, lack any notion in security. Their machines, usually containing unpatched buggy software, are virtually time bombs, waiting to be activated by a malicious attacker. Tools are available, which are eassy to use, and can virtually disrupt the whole network infrastructure of a big company or a government agency. Furthermore, nowadays attackers are often operating like real criminals, since lots of money can be earned in the network disruption business. Competitors, for example may pay huge amounts of money to someone able to spy on, or damage, an industry leader.

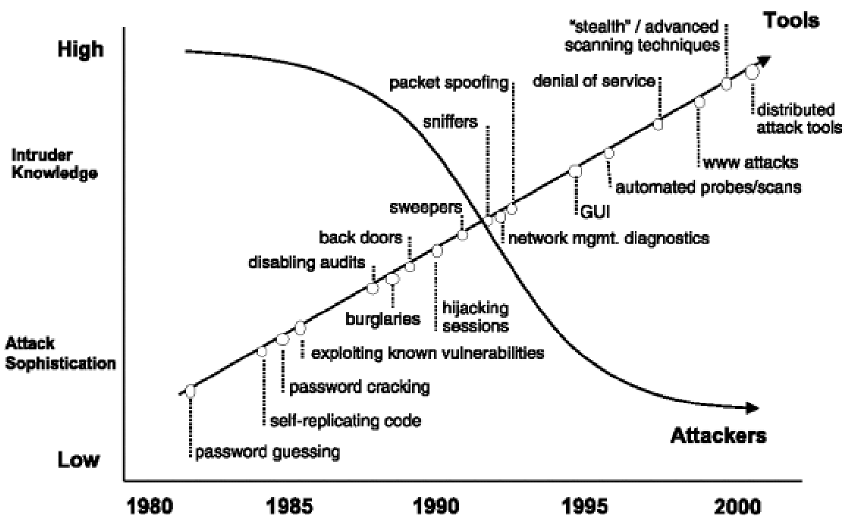


Figure 1.1: Attack Sophistication vs. Attackers' Competences

Consideration like those made so far, show the need for more and more sophisticated techniques to cope with the problem of network security. The serious aspects of network protection, require careful control of any activity on the network. Axelsson [6] showed how serious the impact of detection errors can be, be it due to false alarm, or to missed detection of an ongoing attack. The motivation behind this thesis is, therefore, searching for efficient and effective solutions to the problem of detecting attacks by monitoring network traffic. More precisely, we address the problem of network security by means of pattern recognition techniques. In order to increase the reliability of such techniques, and decrease the number of errors, we propose the employment of multiple classifier systems, able to distinguish between normal and malicious activity by analyzing network traffic. Prior to introducing our contribution, and going into further details about the specific context and the proposed solutions, it's worth providing some background information about the application context.

1.1 Defining Computer and Network Security

In [74], information security is defined as

> *the concepts, techniques, technical measures, and administrative measures used to protect information assets from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, loss, or use.*

Information security is hence the practice of defining and preserving some properties of information which are supposed to be kept unchanged, in order to delivery the originally supposed content to its legitimate users. It is guaranteed by trying to prevent unauthorized access, its disclosure or manipulation. The requirement for information security has evolved through the years, due to the everchanging nature of the environment and of the purpose computers have been used for. Nowadays, it is usually referred to by citing three particular properties of information, namely its *confidentiality*, *integrity* and *availability* [65].

In the early years of informatics, computers were mainly used to perform complicated mathematical operations. Due to the computational limitations characterizing computers in that period, they were merely used as giant, very powerful, yet very complicated to operate and complex calculators. During that period, there were not many computers in the world, and the number of people who could access them was very limited [3]. When computers started being used to store and manipulate important and confidential information, the main concern was to keep the confidentiality level of every bit of information, and to prevent the disclosure of information to people lacking the proper clearance. Hence, at that time information security was all about information *confidentiality*; it was still possible to physically prevent unauthorized users from accessing sensible information.

When the concept of internetworking of computers was introduced [83], keeping information safe became a harder task. Granting remote access to resources improved the chances of leaking sensible information. Also, the number of computers and their users had grown fast, thus making the *computing* community wider, more spread, and less controllable. Furthermore, computers interconnection capability allowed more sophisticated threats to information security. In fact, it became possible to highjack information flows, or to modify the information payload itself, delivering the wrong content to end users. Therefore, information *integrity* was also considered as a fundamental requirement for security.

Recently, computers have gained increasing importance in everyday life. Nowadays, many important services rely on the information storage, management, processing and transmission capabilities of computers and networks. Since crucial information often needs to be retrieved in real time, the dependability of computing and communication infrastructures is considered as an undeniable property. Modern life relies on the *availability* of computational resources, information and communication infrastructures to such an extent, that such a requirement is often considered as one of the most important, and also is one of the most frequently addressed by attackers. Availability is ensured by making a system operational and functional at any given moment, usually through redundancy or accurate resource planning and protection; loss of availability is often referred to as *denial of service*.

Indeed, in this thesis we will address the problem of computer and network security, rather than the more general field of information security. According to Cheswick and Bellovin [20]:

> *computer security is not a goal, it is a means toward a goal: information security.*

A definition of security only taking the three aforementioned properties into account, though, is nowadays considered quite outdated. Other properties, in fact, need to be preserved in order for a system to be secure [80]. Access control has to be preserved, by ensuring that users access only those resources and services that they are entitled to access and that qualified users are not denied access to services that they legitimately expect to receive. The originators of messages or action on the file system must not be able to repudiate their actions. Also, privacy must be taken into account, by ensuring that individuals maintain the right to control what information is collected about them, how it is used, who has used it, who maintains it, and what purpose it is used for.

Besides the previous definition, also a functional definition of computer security can be given. It can be broken into five distinct functional areas [80]:

Risk avoidance A security fundamental that starts with questions like: Does my organization or business engage in activities that are too risky? Do we really need an unrestricted Internet connection? Do we really need to computerize that secure business process? Should we really standardize on a desktop operating system with no access control intrinsics?

Deterrence Reduces the threat to information assets through fear. Can consist of communication strategies designed to impress potential attackers of the likelihood of getting caught.

Prevention The traditional core of computer security. Consists of implementing safeguards. Absolute prevention is theoretical, since there's a vanishing point where additional preventive measures are no longer cost-effective.

Detection Works best in conjunction with preventive measures. When prevention fails, detection should kick in, preferably while there's still time to prevent damage. Includes log-keeping and auditing activities.

Recovery When all else fails, be prepared to pull out backup media and restore from scratch, or cut to backup servers and net connections, or fall back on a disaster recovery facility. Arguably, this function should be attended to before the others.

As stated earlier, and as to the previous points, we are mainly interested in the detection aspect of security.

Also, a definition of security can be based on the type of resource which is controlled [93]:

Physical security Controlling the comings and goings of people and materials; protection against the elements and natural disasters

Operational/procedural security Covering everything from managerial policy decisions to reporting hierarchies

Personnel security Hiring employees, background screening, training, security briefings, monitoring, and handling departures

System security User access and authentication controls, assignment of privilege, maintaining file and filesystem integrity, backups, monitoring processes, log-keeping, and auditing

Network security Protecting network and telecommunications equipment, protecting network servers and transmissions, combatting eavesdropping, controlling access from untrusted networks, firewalls, and detecting intrusions

We will address the problem of network security in the following.
Computer or network security are only a means for preventing improper use of some information. Once again, Cheswick and Bellovin [20] define computer security to be

> *keeping anyone from doing things you do not want them to do to, with, on, or from your computers or any peripheral devices.*

Computers and peripherals are regarded as both attack targets, and instruments that can be used to perform malicious actions. Indeed, such a definition remains very general, since also the three properties of the security paradigm illustrated so far lack a very precise definition. Also, it is difficult to use this definition as is for practical security enforcement.

A more operational definition is presented by Garfinkel and Spafford in their book on Unix and Internet security [94]:

> *A computer is secure if you can depend on it and its software to behave as you expect . . . This concept is often called trust: you trust the system to preserve and protect your data.*

The authors intend for this definition to include natural disasters and buggy software as security concerns, but to exclude software development and testing issues. Actually, as it's claimed by the members of the free software foundation (FSF[1][2]), many software producers and vendors violate this requirement. In the definition of the fundamental freedoms of the software user [114], they claim that what nowadays is called *trusted computing* [90], is by them indeed considered as *treacherous computing* [104], since it doesn't respect the fundamental freedoms of the computer user. Hence, according to the definition of security given in [94], none of the equipment or software conforming to the trusted computing requisite is actually *secure*.

[1]http://www.fsf.org
[2]http://www.gnu.org

1.2 Attack Taxonomies

In [48] the author builds an attack taxonomy starting from two very simple, yet fundamental questions. Such questions regard the potential attack target we are trying to defend, and what type of threat are we defending it from. Usually, attacks try to compromise one of the requisite of information defined in the previous section. Attacks can hence, in general, be classified according to the targeted requisite, and to the means used to violate it. Attackers can exploit many means for pursuing their malicious purposes. These can include the exploitation of flaws in systems project or implementation, faulty hardware, poor access control policies, or bugs in the implementation of the used softwares. In [48] it is stated that, regardless of the cause of a protection failure, there are three and only three sorts of things that can result from a successful attack. First of all, otherwise defect-free information can become corrupt. Second, services that should be available can be denied, and last but not least, information can get to places it should not go [21]. According to Cohen, each of the named events, can be considered as a disruption of information. He explicitly calls those events *corruption*, *denial*, and *leakage* of information. The requisites we discussed so far completely fullfill this definition, which leads to considering information properties such as integrity, availability, and confidentiality for defining the concept of security. Many taxonomies have been proposed in the past, focussing on different aspects of the problem of attack categorization. Some are aimed at giving lists which define all the possible attacks [21] or a list of possible categories. In [20], for example, attacks are classified in the following seven categories:

Stealing passwords methods used to obtain other users' passwords;

Social engineering talking your way into information that you should not have;

Bugs and backdoors taking advantage of systems that do not meet their specifications, or replacing software with compromised versions;

Authentication failures defeating of mechanisms used for authentication;

Protocol failures protocols themselves are improperly designed or implemented;

Information leakage using systems such as finger or the DNS to obtain information that is necessary to administrators and the proper operation of the network, but could also be used by attackers;

Denial-of-service efforts to prevent users from being able to use their systems.

By observing this taxonomy, it is clear that it can possibly include almost every type of known attack. Indeed, it is often confusing, since categories represent either the target of the attack, its final purpose, or the strategy used to obtain the desired effect. Also, large lists of items are rarely fully comprehensive, since it's difficult to keep up with the pace at which different types of novel attacks are introduced.

In an effort to overcome taxonomies based on mere listings, Stallings proposed an attack categorization based on modification to the information flow [103] resulting from the attacker's actions. Given the model of normal information exchange between two entities, as represented in figure 1.2, he individuates four possible modification of such flow. In Stalling's taxonomy,

Information Source
Information Destination

Figure 1.2: Normal Information Flow

a distinction is made between active and passive attacks. Passive attacks are those which don't interfere with the normal information flow. What they do is just collect the information, without any effect on its normal transmission.

Interception (figure 1.3(b)) is viewed as a passive attack, since it only intercepts the information flow, directing it towards a collecting node. The natural information flow is preserved unmodified. On the other hand, interruption (figure 1.3(a)), modification (figure 1.3(c)) and fabrication (figure 1.3(d)) are viewed as active attacks. This taxonomy is particularly interesting since it expresses the properties of attacks as processes, individuating their algorithmic nature, focussing on the general problem of information exchange, and trying to find common properties in their life cycle.

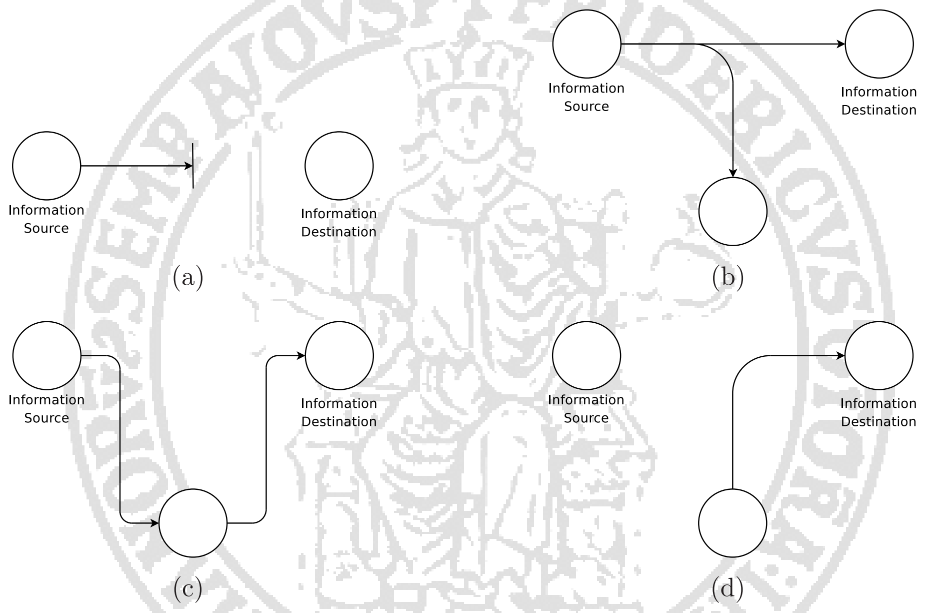


Figure 1.3: Four ways to describe attacks as information flow modification: (a) Information Flow Interruption; (b) Information Flow Interception; (c) Information Flow Modification; (d) Information Flow Fabrication.

Howard instead gives a more operational view of attacks [48]. He groups attacks according to the type of attacker, the implemented access strategy, the tools used and the expected results.
In [47] a taxonomy of threats to computer and network security is presented. According to previous well known taxonomies, Hansman and Hunt introduce a number of properties which a good taxonomy should have, such as:

Accepted [3, 48] The taxonomy should be structured so that it can be become generally approved.

Comprehensible [66] A comprehensible taxonomy will be able to be understood by those who are in the security field, as well as those who only have an interest in it.

Completeness [3]/**Exhaustive** [48, 66] For a taxonomy to be complete/exhaustive, it should account for all possible attacks and provide categories for them. While it is hard to prove a taxonomy is complete or exhaustive, they can be justified through the successful categorization of actual attacks.

Determinism [53] The procedure of classifying must be clearly defined.

Mutually exclusive [48, 66] A mutually exclusive taxonomy will categorise each attack into, at most, one category.

Repeatable [48, 53] Classifications should be repeatable.

Terminology complying with established security terminology [66] Existing terminology should be used in the taxonomy so as to avoid confusion and to build on previous knowledge.

Terms well defined [15] There should be no confusion as to what a term means.

Unambiguous[48, 66] Each category of the taxonomy must be clearly defined so that there is no ambiguity as to where an attack should be classified.

Useful[48, 66] A useful taxonomy will be able to be used in the security industry. For example, the taxonomy should be able to be used by incident response teams.

These properties are the result of an accurate study of previous models, and try to respect all the good accomplishments of earlier research in the field of attack categorization.

Hansman and Hunt propose to use a somewhat *geometrical* framework in order to identify attack types. They propose a structure using four dimensions. The first dimension (table 1.1), also referred to as base dimension, categorises attacks according to the attack vector. The second dimension (ta-

Viruses	File Infectors System/Boot Record Infector Macro	
Worms	Mass Mailing Network Aware	
Trojans	Logic Bombs	
Buffer Overflows	Stack Heap	
Denial of Service Attacks	Host Based	Resource Hogs Crashers
	Network Based	TCP Flooding UDP Flooding ICMP Flooding
	Distributed	
Network Attacks	Spoofing	
	Session Hijacking	
	Wireless Attacks	WEP Cracking
	Web Application Attacks	Cross Site Scripting Parameter Tampering Cookie Poisoning Database Attacks Hidden Field Manipulation
Physical Attacks	Basic	
	Energy Weapon	HERF LERF EMP
	Van Eck	
Password Attacks	Guessing	Brute Force Dictionary Attack
	Exploiting Implementation	
Information Gathering Attacks	Sniffing Mapping Security Scanning	Packet Sniffing

Table 1.1: Attack Taxonomy [47] – First Dimension

ble 1.2 on the following page) covers the attack target. Classification can either be very fine-grained, down to the program version number affected by a specific vulnerability, or cover a class of targets, such as entire operating systems' processes. The third dimension covers the vulnerabilities and exploits, if they exist, that the attack uses. There is no structured classification for vulnerabilities and exploits due to their possible infinite number. The fourth dimension takes into account the possibility for a side effect for a specific attack. Some attacks, like trojan horses for example, carry the burden of what's hidden in the *horse*, which is usually the most dangerous part of it. In the following, we will mainly deal with network security issues. The approach we propose aims at detecting anomalous activities within a network by analyzing network traffic. Such traffic characteristics are tightly related to activities on single hosts, and can reflect the presence of ongoing mali-

Hardware	Computer	Hard-disks ...			
	Network Equipment	Hub Cabling ...			
	Peripheral Devices	Monitor Keyboard ...			
Software	Operating System	Windows Family	Windows XP Windows 2003 Server ...		
		UNIX Family	Linux	2.2 2.4 ...	
			FreeBSD	4.8 5.1 ...	
			...		
		MacOS Family	MacOS X	10.1 10.2 ...	
		...			
	Application	Server	Database		
			Email		
			Web	IIS	4.0 5.0 ...
				...	
			...		
		User	Word Processor	MS Word	2000 XP ...
				...	
			Email Client	...	
			...		
Network	Protocols	Network Layer	IP ...		
		Transport Layer	TCP UDP ...		
		...			

Table 1.2: Attack Taxonomy [47] – Second Dimension

cious activities. As a matter of fact, the network nowadays is probably the most commonly used means of spreading malicious code and computer infections, and perpetrating attacks. Referring to the last cited taxonomy, we will mainly deal with Worms, Network attacks and both distributed and network based denial of service attacks. We will mainly focus on the effects these types of attacks have on networks and on network traffic properties, and what are the observed symptoms as to network protocol analysis.

1.3 Tools for Network Security

As to the previous section, we are going to address mainly the problem of network based attacks, by analyzing properties of network traffic. There are many security tools which can address each of the specific types of attacks presented in every attack taxonomy. The most popular for example are antiviruses, malware detectors, unpackers, antispyware and personal firewalls. All those tools are meant to monitor and protect a single host.

In the framework of this thesis, we are interested in studying the problem of network based detection of attacks targeting both the components of the network and the protocols which allow them to interoperate. In the following there will be a description of the main approaches to such a problem. We distinguish three main phases in network protection [41]. Many properties and requisites can be defined for each phase [77]. Intrusion detection aims at defining techniques which allow to detect attacks while they are being performed. Intrusion prevention aims at defining strategies and policies which can prevent intrusion from happening or succeeding, or at least reduce the probability of such events. Intrusion reaction instead is about methods and techniques which allow, as suggested by the name, a proper reaction to an attack. It involves forensic analysis, and attack traceback, aimed at locating the attacker and preventing him from causing further damage. It also involves restoration techniques to recover from the damages resulting from a successful attack.

In [102] Specht et al. introduce a comprehensive approach to attack detection and reaction, by focussing on the case of Distributed Denial of Service Attacks. First of all, it's necessary to prevent the attack from being performed, by avoiding the creation of the necessary preconditions. Once an attack is successfully performed, if it's detected, it's necessary to do anything possible to prevent the same attack from succeeding again. By controlling the protected network, it's necessary to eliminate all the possible attack handles. Also, a clever activity logging strategy can help for forensics, and future attack avoidance.

1.3.1 Intrusion Detection

An Intrusion Detection System (IDS) analyzes a data source and, after preprocessing the input, lets a detection engine decide, based on a set of classification criteria, whether the analyzed input instance is normal or anomalous,

given a suitable behavior model.
According to Fuchseberger [41] intrusion detection systems can be of four types:

Behavior based statistical techniques are used to detect penetrations and attacks; these begin by establishing base-line statistical behavior, and then measuring the deviation from the base-line.

Knowledge based look for attack signature either in network traffic or system logs; also known as misuse based intrusion detection.

Host based derive from mere log file analyzers, and are designed as host based applications running in the background of presumed critical, sensitive hosts; detect attack patterns that can only or easier to be found on a host level basis.

Network based monitor network traffic at packet level; can be either centralized, or made of multiple distributed monitoring stations.

Yet, this model is somewhat ambiguous. In fact, the four types of IDS are not defined according to a single property. The first two types, in fact, are identified by the type of approach to detection, whereas the latter are identified in terms of the analyzed data source.

Debar et al. [45] (figure 1.4 on the next page) also consider the usage frequency of the intrusion detection process. It can be either continuous, or periodic. Continuous analysis allows to collect information about everything happening on the monitored source of information. Yet, in particular scenarios, such as high speed network monitoring, a controlled sampling of analyzed information can result in better performance, by performing controlled, rather than aleatory, packet and information loss [101].

More in general, intrusion detection systems can be grouped according to several properties. As to the analyzed data source, they can be grouped into two main categories: *Network-based Intrusion Detection Systems* (N-IDS) [110], and *Host-based Intrusion Detection Systems* (H-IDS) [4, 109]

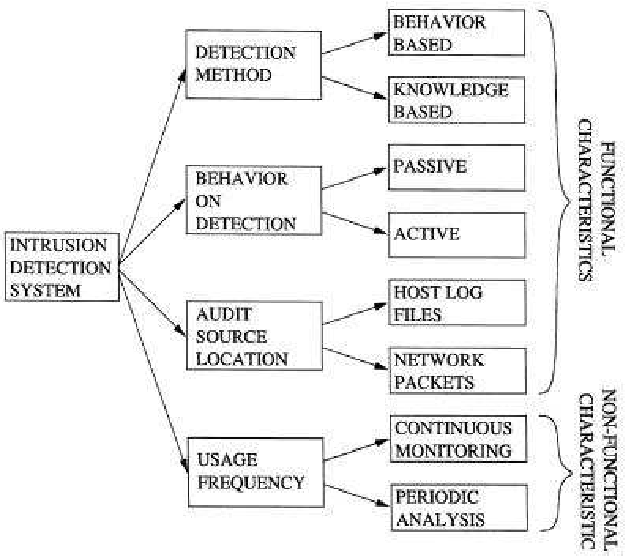


Figure 1.4: A taxonomy of intrusion detection systems [45]

This classification depends on the information sources analyzed to detect an intrusive activity. N-IDS analyze packets captured directly from the network. By setting network cards in promiscuous mode, an N-IDS can monitor traffic in order to protect all of the hosts connected to a specified network segment. On the other hand, H-IDS focus on a single host's activity: the system protects such a host by directly analyzing the audit trails or system logs produced by the host's operating system. Intrusion Detection Systems can be roughly classified (Figure 1.5 on page 19) as belonging to two main groups, also depending on the employed detection technique: *anomaly detection* and *misuse detection*[8]. Both such techniques rely on the existence of a reliable characterization either of what is *normal* and what is not, in a particular networking scenario.

Anomaly Detection

More precisely, anomaly detection techniques base their evaluations on a model of what is normal, and classify as anomalous all the events that fall outside such a model [63]. Indeed, if an anomalous behavior is recognized, this does not necessarily imply that an attack activity has occurred: only few anomalies can be actually classified as attempts to compromise the security of the system. Thus, a relatively serious problem exists with anomaly detection techniques which generate a great amount of false alarms. On the other side, the primary advantage of anomaly detection is its intrinsic capability to discover novel attack types. Numerous approaches exist which determine the variation of an observed behavior from a normal one. A first approach is based on statistical techniques. The detector observes the activity of a subject (e.g. number of open files or TCP state transitions), and creates a profile representing its behavior. Every such profile is a set of "anomaly measures". Statistical techniques can then be used to extract a scalar measure representing the overall anomaly level of the current behavior. The profile measure is thus compared with a threshold value to determine whether the examined behavior is anomalous or not. A second approach, named *predictive pattern generation*, is based on the assumption that an attack is characterized by a specific sequence, i.e. a *pattern*, of events. Hence, if a set of time-based rules describing the temporal evolution of the user's *normal* activity exists, an anomalous behavior is detected in case the observed sequence of events significantly differs from a normal pattern.

Misuse Detection

Misuse detection is performed by accurately describing all the possible unwanted behaviors that need to be detected [85, 54, 76]. Hence, all the traffic patterns conforming to such a behavior description are classified as attacks. A special case of misuse detection is represented by the so called *signature detection* [13, 107] technique. Such a technique is based on the assumption that an intrusive activity is characterized by a precise and often well recognizable signature, i.e. a well-known pattern. Such signatures are usually represented by specific configurations of packet header fields, of specific payload contents.

Similarly to anomaly detection, misuse detection can use either statistical techniques or even a neural network approach to predict intrusions. Indeed, the rule-based approach is the most used to detect an attack (SNORT[3][13] and Bro[4][82]). Possible intrusive behaviors are coded by means of a set of rules [111]: as soon as the examined event matches one of the rules, an attack is detected. A drawback of this approach is that only well-known unwanted activities can be detected, so that the system is vulnerable to novel aggressions; sometimes, few variations in an attack pattern may generate an intrusion that the IDS is not able to detect.

The main problem related to both anomaly and misuse detection techniques resides in the encoded models, which define normal or malicious behaviors. Although some recent open source IDS, such as SNORT or Bro, provide mechanisms to write new rules that extend the detection ability of the system, such rules are usually hand-coded by a security administrator, representing a weakness in the definition of new normal or malicious behaviors. Recently, many research groups have focused their attention on the definition of systems able to automatically build a set of models. Pattern recognition techniques are frequently applied to audit data in order to compute specific behavior models (MADAM ID [64], ADAM [9]).

Axelsson [7, 6] also proposes to use the time of detection, the granularity of data-processing, response to detected intrusions (distinguished between active and passive), the locus of data-processing and collection, the intrinsic security and the degree of interoperability to categorize intrusion detection systems.

1.3.2 Intrusion Prevention

Intrusion prevention systems were introduced in order to solve the typical problems of both passive monitoring security systems and firewalls [113]. According to [116] intrusion prevention systems map traffic classification and

[3]http://www.snort.org

[4]http://www.bro-ids.org

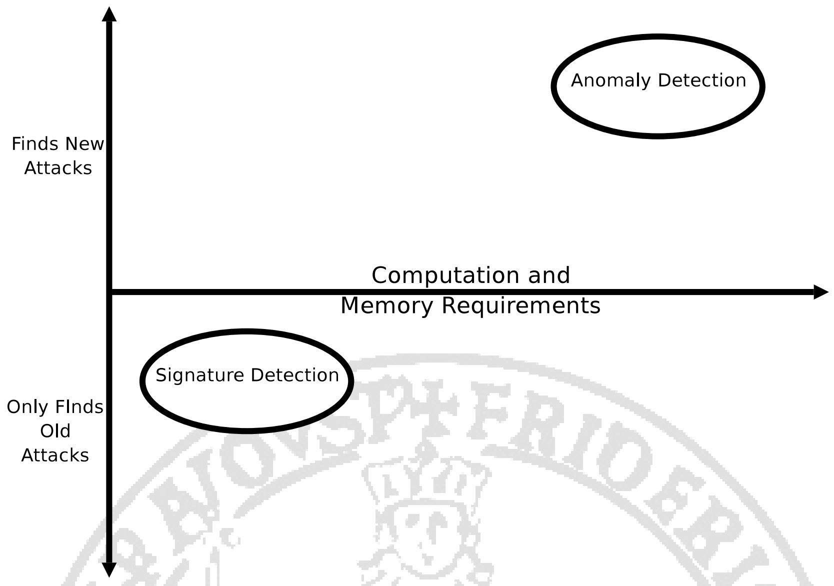


Figure 1.5: Approaches to Intrusion Detection

event detection to responses. It monitors network on the fly. An intrusion detection module classifies the traffic, and two modules called the policy moderator and enforcer apply the selected policy according to the detected event.
The policy moderator receives as input flow specifications, state, and context from the traffic classifier, firewall policies such as PASS, BLOCK, PROXY, MONITOR, and events detected by the detection engine. It outputs the intrusion alerts or a more general indication of the intrusion type, obtained by an aggregate of intrusion alerts. The policy enforcer is entitled to the actual enforcing of the policy. It performs an alert to response mapping, and executes traffic enforcement actions, such as:

Interface Actions random packet drop, rate limiting, and interface blocking

Network Actions packet or connection blocking, IP or URL blocking

Service Actions TCP or UDP Service (Telnet, FTP, HTTP , SNMP) filtering, NAT and port forwarding.

File Actions email attachment stripping, http or FTP file blocking

Content Actions protocol command blocking, HTTP content filtering, etc.

All the traffic enforcement actions are triggered by an activation condition, and stopped by a deactivating condition. An attack response matrix is defined, which efficiently describes the correspondence between detected attack and associated action. In [41] intrusion prevention systems are distinguished between rate-based and content-based. Rate based systems take action on traffic based on the network load, measured in terms of either packet, byte or in general activity rate. The main disadvantage of this type of systems is the exact definition of what type of load should actually trigger a reaction, without risking unwanted traffic-blocking actions. Content based systems, instead, take action against traffic by searching for signatures inside packets. The drawbacks of such systems are the same as any other signature based system.

1.3.3 Intrusion Reaction

In [79] the authors propose an interesting survey on intrusion response techniques (figure 1.6 on the next page). They classify such systems according to both the type of action at the occurrence of a detection, and the degree of automation. Response systems can be either active and passive when responding to a detected attack. The former can modify some network configuration parameters which affect traffic characteristics, whereas the latter may only report the detection and don't take any active action on network traffic. As to the degree of automation in response, the analyzed systems can either notify the detected anomalous event, or react to it. The reaction can either be manual, by the system operator itself, or automated. Automated response systems can be characterized according to their ability to adjust both the detection parameters and the reaction intensity according to the context. They can be programmed to respond in a timely manner, or to delay the response, in order to collect further information about the un-

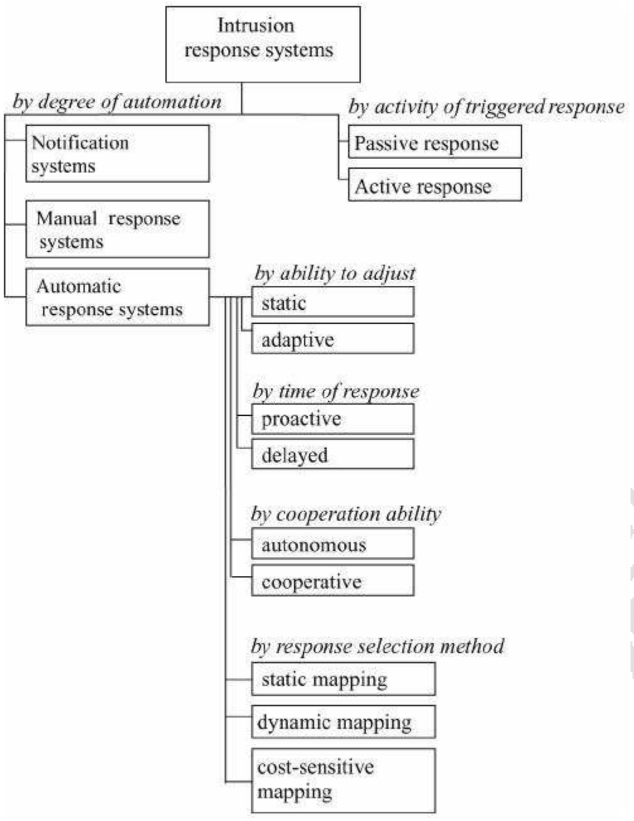


Figure 1.6: A Taxonomy of Intrusion Response Systems

suspecting attacker, and eventually react more effectively. The cooperation ability is also taken into account. Since networks are complex distributed entities, a good degree of cooperation between the security components can be useful to have a better reaction to a critical situation. A proper reaction to attacks can also involve the restoration of the last working condition of the system, or the traceback to the attack source. By accurately logging malicious activities, both these tasks can be accomplished [96, 106, 84].

1.4 Thesis Outline

After briefly introducing the operational context of this thesis, in this section we will give a synthetic outline of the rest of the work. In chapter 2 on page 23, we will introduce some notions on classification theory, and multiple classifier systems. We will point out a meeting point between the problem of network security, and the general problem of classification. Hence, we'll present a reference model, which allows us to employ classifiers for network traffic classification.

Since some of the classifiers commonly used for network security need a properly labeled dataset for training, the need for commonly acknowledged and up-to-date datasets is always urging. In chapter 3 on page 53, we will tackle the problem of data collection for training supervised classification systems. Once the problem of user privacy is introduced, we will propose a solution for sensible data anonymization, and then show how to automate the process of packet labelling by means of a multiple classifier system. We will define an algorithm for iterative incremental creation of labelled traffic datasets, and show how supervised classification systems perform on a dataset built this way.

In chapter 4 on page 95, we will present an architecture for the actual deployment and management of multiple classifiers within a real network scenario. This is a necessary preliminary step for the exploitation of multiple classifier systems for intrusion detection. Hence, we will present some system which, by using several strategies for information combination, are able to improve the performance of several base classifiers, joined together in multiple classifier systems.

In chapter 5 on page 119 some conclusions are drawn. Our contribution is pointed out, referring to the previously described work, and some directions and proposals for future works are proposed.

Chapter 2

Classifiers for Network security

Among the possible countermeasures to computer and network attacks, we are interested in studying network based Intrusion Detection Systems [7]. In this work we want to study the problem of monitoring the network and its users' behavior in order to detect and report suspicious events. To some extent, the problem of intrusion detection can be regarded as intermediate among the prevention and reaction phase. In fact, it proves necessary in case the prevention policies fail, and constitutes a preliminary step to reaction.

By monitoring network segments, and observing some significant traffic properties, we aim at being able to infer whether any anomalous activity is going on. Hence, there are several aspects of the problem to deal with. First of all, we need to find a model which is appropriate to synthesize traffic properties which allow to distinguish between acceptable and anomalous behaviors. There's a wide choice of approaches to traffic analysis, which often borrow techniques from several research fields, such as pattern matching, statistics and artificial intelligence. We propose solutions based on techniques coming from the field of pattern recognition and artificial intelligence.

Once a way to describe traffic properties is chosen, we need to define how to practically transform "network traffic" in the selected representation. Since we're dealing with network security, the necessary steps involve traffic sniffing. Starting from raw packets, the value of parameters describing interesting traffic properties must be calculated. Due to the inherent complexity of network activity, and to its distributed nature, we need to define a strategy to effectively sniff the traffic. We'll present our proposal for traffic sniffing, and also its evolution toward a distributed network monitoring architecture, which takes the problem of multiple, spread monitoring sensors, into account. We'll describe the process of calculating some specified models, starting from raw network traffic. Then, we'll how how, among other techniques, classifiers can be practically used for network security.

2.1 Classification theory

As the name itself suggests, classification theory is about solving the problem of assigning real life entities to one out of a set of categories. Such a general definition indicates how flexible the theory of classification is, and inherently suggests its numerous application fields. Some authors [86] date the first examples of classification theory and pattern recognition back to ancient Greece [16, 5], when scientists and philosophers were wondering about the inherent properties of natural phenomena, in order to give explanations for them. Without going so far away in time, we can find a number of applications for classification theory in nowadays world. Classifiers are used every time an automatic system has to decide whether an entity it is observing belongs to one out of the categories it knows. Significant applications today include, but are not limited to, video surveillance [24], optical character recognition [39], handwriting recognition [40] and video segmentation [95]. Classification theory is also often used as a support for diagnosis in medicine [58] and in general in shape recognition problems [27]. Though the core of classification is very general and usually independent of the application context, there's a huge work around it to be done. In fact, though such techniques can often be reused, with satisfactory results, in several different fields, there's a considerable amount of work and study in finding a representation of each problem, which is suitable with the application of such a flexible instrument. Real life problems are inherently complex, and usually don't have simple representations. Despite the enormous amount of computation resources available nowadays, at some point problems have to be reduced in terms of discrete representations in order to be manipulated by calculators.

2.1.1 Feature extraction and selection

As stated before, in order to actually use a classifier for solving real problems, the problem itself needs a suitable representation. Such a representation requires the definition of the possible categories which have to be recognized, and also the description of the entities to classify in terms of a certain number of parameters. Such parameters are usually referred to as *features* [51]. Features are usually represented in arrays, and can be distinguished according to the type of value they can assume. They are usually grouped into two sets, as depicted in figure 2.1: *quantitative* features and *qualitative* features. The former can have both discrete and continuous values, whereas the latter can be of ordinal or nominal type. Both types of features can be combined

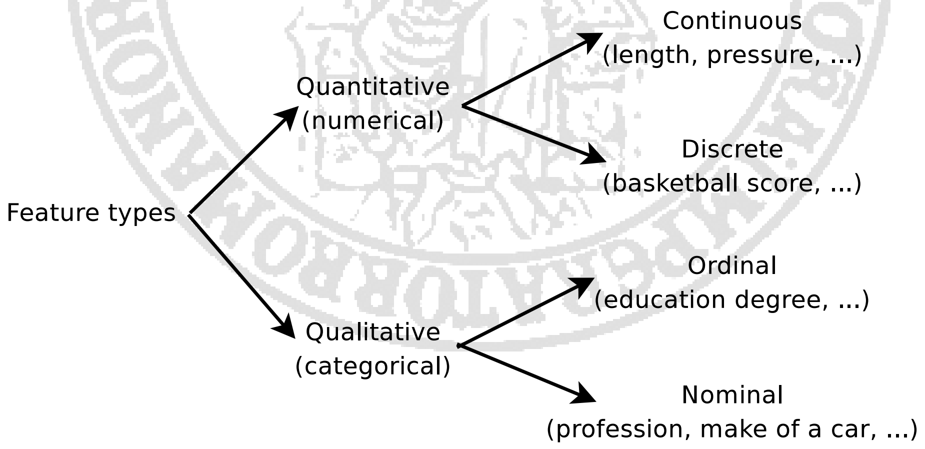

Feature types
Quantitative (numerical)
Qualitative (categorical)
Continuous (length, pressure, ...)
Discrete (basketball score, ...)
Ordinal (education degree, ...)
Nominal (profession, make of a car, ...)

Figure 2.1: Types of features

in the description of a particular phenomenon. Several methods have been proposed for the discretization of continuous features, such as the ones described in [30]; in [71] the impact of such a discretization process has been estimated. Feature definition can be deemed as the most critical phase in the project of a classification system. In fact, the performance of a classifier is heavily influenced by the features chosen for entity representation. Usually, the choice of representative features is performed by an expert of the problem under exam. Since features are usually represented as arrays of parameters, we can imagine that features represent points in a vector space. In order to allow this type of representation, a conversion from qualitative to quantitative features is required. Indeed, such a process is usually heuristic and very subjective. By bearing this in mind, features must be chosen in order to allow a good separability of points belonging to different categories. Therefore, when designing features for a new problem, a number of properties has to be selected, which make the job of the classifier as easy as possible. Some overlap between sets representing different categories may occur, thus generating unrecoverable classification errors. The job of the classifier consists in defining functions describing the boundaries which separate regions of the feature space containing points of different categories. The most simple case, for example, is the case of linear separability. Such a property occurs when points of the vector space belonging to different categories can be separated by means of hyperplanes.

Even though a large number of features might be deemed useful for some particular classification problem, it doesn't mean that, for every instance of the problem, all of those features will be of real use. In some cases, for example, the value of a particular feature might be subject to high noise, or might be unmeasurable due to environmental constraints. Therefore, a feature selection process is usually employed [81, 67]. The purpose of feature selection is to select the features granting the highest discriminating power among classes on a given training set (this term will be clarified in section 2.1.2 on the next page). On the other hand, the process of feature extraction [89, 108] aims at calculating a kernel for a subset of the original feature space. The new subspace will span a smaller dimension, and the base vectors will be different, since they are obtained by elaborating the original features. Since we

want to preserve the original meaning of assigned features, we will mainly deal with feature selection rather than extraction.
Several tools performing both operations are publicly available[1].

Other forms of representation for classification are also proposed in [31]. The authors propose to represent object in terms of relations to other objects, instead of using numerical models such as features. This work aims at bridging the gap between statistical and syntactic pattern recognition, by completely overcoming the representation differences which oppose the two approaches.

Normalization

The process of normalization enables to disregard environmental influence on features value. Due to different observation condition, the value of a feature could change, even though such a change doesn't reflect a different property. As an example, when building a system recognizing geometrical shapes, the orientation of one of the pieces to classify may influence some observed properties, such as angles, straight and curved lines. With the normalization process, it is possible to compensate the effect of such predictable environmental factors influencing feature measurements.

2.1.2 Supervised vs. Unsupervised systems

Once the most discriminating features have been selected, it's time to deal with the choice of the best suited classification technique. A method to discriminate among different classifiers, relies on the type of training strategy they exploit. According to such criterion, classifiers can be roughly classified in *supervised* and *unsupervised* [62, 59].

Supervised Systems Supervised systems rely on the existence of a *teacher* helping the system in building its own knowledge about the problem. Usually,

[1]Tooldiag – http://www.inf.ufes.br/~thomas/home/tooldiag.html

such a teacher provides the classifier with a previously labelled training set. By analyzing labelled instances of training data, the classifier is able to define the decision boundaries which will be later used during the operating phase.

Unsupervised Systems Unsupervised systems, instead, perform a sort of clustering operation. Due to the absent of an explicit teaching phase, unsupervised systems try to group together entities which share common properties in the feature space. As an example, such techniques might group together densely populated regions of the feature space, assuming that all the entities represented by points lying in such regions share common properties in the real world. Obviously, different clustering techniques might lead to different results. Often the user is given the chance to set the a-priori estimated number of possible different clusters, to impose a sort of constraint on the system.

Some classification techniques, such as Support Vector Machines (SVM), can be used in both supervised and unsupervised fashion [18, 50]. Recently, this distinction between two types of classification techniques has become more loose; in fact, intermediate levels of supervision have been introduced between total supervision and the total absence of it. Semi-supervised techniques, for example, make use of both labelled and unlabeled data for training. It has been proved that the combined use of both types of data can significantly improve detection performance [120].

2.1.3 Error evaluation and decision cost

An important role in classification theory is played by error evaluation. Given a labelled dataset, the most straightforward strategy for evaluating the performance of a classification system is just counting the number of committed errors. Often, the relative amount of errors is given, with respect to the total number of analyzed samples. Yet, it's worth going into further details in understanding the nature of each error. In fact, it's often important to understand how many times a sample from a category, or a class, has been mistaken as belonging to a different class. The reason why this is helpful, is that not all errors are equally serious. Depending on the nature of the real problem, some errors might lead to worst consequences than others. In order to express the percentage of errors in terms of confusion among classes, the *confusion matrix* is used. Confusion matrices represent, for each given real class, the number of times it's been mistaken for any other class. An example for the general n classes case is depicted in table 2.1.

	Assigned Class			
True Class	$\hat{\omega}_1$	$\hat{\omega}_2$	$\dots$	$\hat{\omega}_n$
ω_1	c_{11}	c_{12}	$\dots$	c_{1n}
ω_2	c_{21}	c_{22}	$\dots$	c_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
ω_n	c_{n1}	c_{n2}	$\dots$	c_{nn}

Table 2.1: Confusion matrix for n classes classification

In the case of one class classification, the problem of classification is simply reduced to recognize whether a specific sample belongs to the considered class. If this is not the case, the sample is simply *not* assigned to the class of interest. The problem can be formally represented by naming two possible classification outcomes, namely *Positive* and *Negative*, representing the only two possible options taken into account. In fact, in such a case, the occurrence of a particular class is searched for. Anything outside such a class is tagged as *Negative*. The corresponding confusion matrix is represented by table 2.2.

	Assigned Class	
True Class	$\hat{P}$	$\hat{N}$
P	TP	FN
N	FP	TN

Table 2.2: Confusion matrix for one class classification

In such a case, the elements of the confusion matrix are named, respectively, *True Positives* (TP), *False Negatives* (FN), *False Positives* (FP) and *True Negative* (TN). Such quantities can also be expressed as relative to the total amount of patterns or samples belonging to either the class of interest, or not belonging to it.

2.2 Multiple Classifier Systems

When using classification techniques, a critical point is reached, where not many improvements can be obtained in classifiers performance. The employment of multiple classifiers, therefore, is justified in order to push such critical point further, and improve saturated performance of a set of base classifiers. Assuming we have a number of base classifiers, characterized by a certain degree of training and accuracy in classification, each of them will have a specified generalization power. By using an ensemble of multiple classifiers, the expectation is to obtain a better overall generalization error. In fact, it may be possible to compensate the generalization error of a classifier by means, eventually, of good generalization properties of other classifiers on the same sample. In [29] three reasons are suggested why a multiple classifier system may perform better than a single classifier. First of all, the aforementioned generalization issue is an advantage for multiple classifiers. Given the probability of a single classifier committing a generalization error, the chances of an ensemble of classifiers all committing the same error at once are much lower. Furthermore, since many classification techniques use, for computational reasons, some suboptimal optimization strategies, their results might be suboptimal as well. By using multiple classifiers, such effects can be compensated by randomly exploring several suboptimal training patterns, and hence the resulting multiple classifier might perform better. Multiple classifiers can also overcome the representational issue of single classifiers. For example, let's assume linear classifiers are used. The representational power of decision boundaries by these classifiers is inherently limited. Indeed, by combining a number of linear functions, it's possible to describe decision boundaries as complicated as necessary, to best fit the problem at hand.

2.2.1 Strategies for Combining Multiple Classifiers

When using multiple classifiers, it's possible to manipulate several components of the system, in order to pursue the desired performance level. In [56] four approaches are individuated, as depicted in figure 2.2 on the following page. First of all, it's obviously possible to choose among several combination techniques. Such a choice can be made, for example, between supervised or unsupervised combinator, according to problem requirements. Also, combination rules might change with respect to the data to combine, that is to say to different types of output by the base classifiers.
The choice of base classifiers may also affect the performance of the multiple classifier system. It's possible to choose base classifiers according to their expected performance, or to the problem domain. Actually, there is little evidence which justify the choice of a uniform ensemble of classifier rather than an etherogeneous one. This choice is tightly related to the features used to model the problem under exam. Each classifier can receive as input a different subset of features, thus making the choice of similar techniques to implement base classifiers reasonable. In such a case, in fact, by working on different features, all the classifiers will be *different*, though implementing the same technique. Also, each classifier may use a different portion of the dataset under exam. All these degrees of freedom reflect the will to combine the classification outcomes of different classifiers, having several different points of view about the analyzed problem. The intuition is that, by combining *experiences* built on different backgrounds, be it due to differences in data and/or implemented learning algorithms, a better understanding of the studied phenomenon can be built overall.

2.2.2 Fusion vs. Selection

The combination approaches proposed so far can be roughly divided between fusion and selection strategies. The former aim at finding the best way, given a set of base classifiers, to combine them in order to obtain the expected results. Approaches based on fusion can rely on average, majority voting or

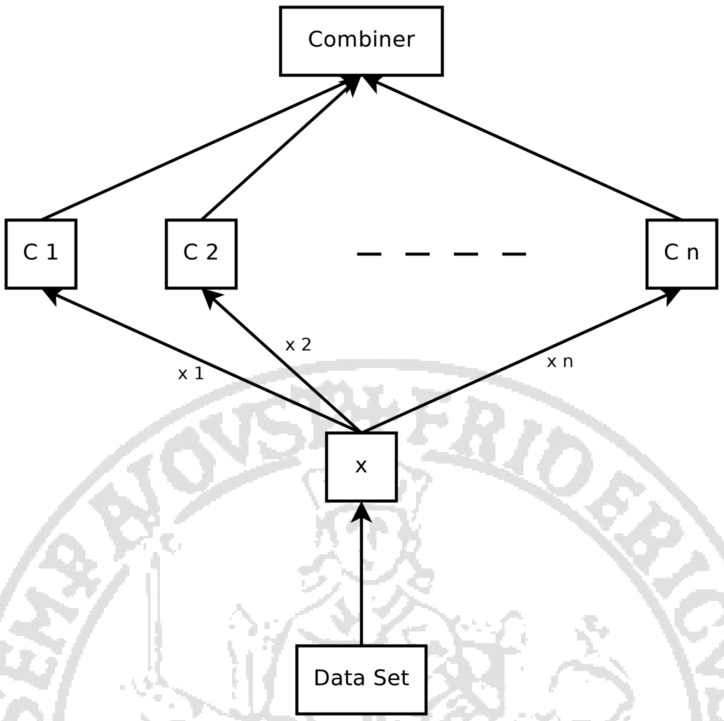


Figure 2.2: Approaches to building classifier ensembles

other techniques, and require a means of taking into account the different relative importance of each base classifier, and eventually its reliability. Also the cost of each decision may influence such combination strategies. When using selection, instead, classifiers outcomes are not combined, but the best performing classifier is selected at the occurrence of every pattern to classify.

2.2.3 Decision vs. Coverage Optimization

When using multiple classifiers, there are two approaches regarding base classifiers. Decision optimization can be regarded as a superset of combiners based on fusion or selection. It refers to methods which allow to choose and optimize the combiner for a given ensemble of base classifiers. Coverage optimization, instead, refers to methods for creating diverse base classifiers assuming a combiner has been selected. We will mainly follow this second approach in the following. Yet, we will select base classifiers, instead of creating them on purpose.

2.2.4 Trainable vs. Non-trainable Ensembles

The difference here is straightforward. Some combiners do not need training after the classifiers in the ensemble have been trained individually. An example is the majority vote combiner, which we will use in section 4.1 on page 95 or 3.5 on page 73. Other combiners, such as the one presented in 4.2.4 on page 107 need additional training. Another class of combiners doesn't need prior training, but develop its training during the training of base classifiers, adapting itself to their evolution.

In [115] three types of classifiers are individuated, with respect to the type of their output. Obviously, different types of output require different adaptations for combination, and are characterized by both different drawbacks and advantages.

Type 1 (The Abstract level). Each classifier produces a class label Thus, for any object to be classified, the classifier outputs define a vector. At the abstract level, there is no information about the certainty of the guessed labels, nor are any alternative labels suggested. By definition, any classifier is capable of producing a label for the object, so the abstract level is the most universal one.

Type 2 (The Rank level). The output of each classifier is a subset of the labels, with the alternatives ranked in order of plausibility of being the correct label. Especially suitable for problems with a large number of classes.

Type 3 (The Measurement level). Each classifier produces a vector. Each of its elements represents the support for the hypothesis that such a vector is related to a sample from the corresponding class.

This section aims at briefly introducing multiple classifier systems. Further details will be provided, on a case by case basis, throughout the thesis.

When necessary, the used combination techniques will be described more in depth.

2.3 Network Security as a Classification Problem

In the previous sections of this thesis, we've been referring to traffic classification and attack detection problems. It is worth pointing out the contact point between the two fields, in order to both clarify and justify the "abuse" of terms we've committed so far, and to explain how such research fields meet [43]. It is evident that, by aiming to distinguish among several types of user behaviors on the network, we want to be able to divide them into categories. Let's assume we are able to identify a number of equivalence classes of network traffic, described by a specified number of properties of interest. By observing each user's behavior, we want to be able to decide to assign it to one out of all the possible categories. By doing this, we implicitly assume the knowledge and, most important, the existence of only a predefined number of such categories. Furthermore, we assume to be able to find a reduced set of properties, to assign each of the entity we want to classify to the appropriate class.
In practice, in order to treat network traffic classification for security purposes as a classification problem, several steps are required. First of all, a model for the problem has to be defined. In the following, we'll deal with the classification of connections. In our specific case, we only consider TCP, UDP and ICMP traffic. In the case of TCP traffic, the definition of connection naturally comes from protocol specification, since TCP is a connection oriented protocol. In the case of UDP and ICMP, instead, we treat each packet as a connection itself, since both protocols have a connectionless nature. Hence, we just respect the protocol specification, without forcing any structure over the native protocols. Such structure, in fact, would require the definition of connection timeouts and other system parameters, which

could affect the obtained results, and constitute an unwanted degree of freedom for problem analysis.
Whenever a new packet is sniffed on the networks, it is assigned to its corresponding connection. Each packet is considered as an uptodate to the connection status, therefore all the traffic model parameters related to such connection are updated.
In the following we present the reference model we used to employ classifiers for network security. We'll illustrate how feasible feature extraction from live traffic is. The presented model is meant to extract the features defined in [64]. We'll describe some implementation details, and the performance overhead due to feature computation in a particular network scenario.

2.3.1 The Reference Model

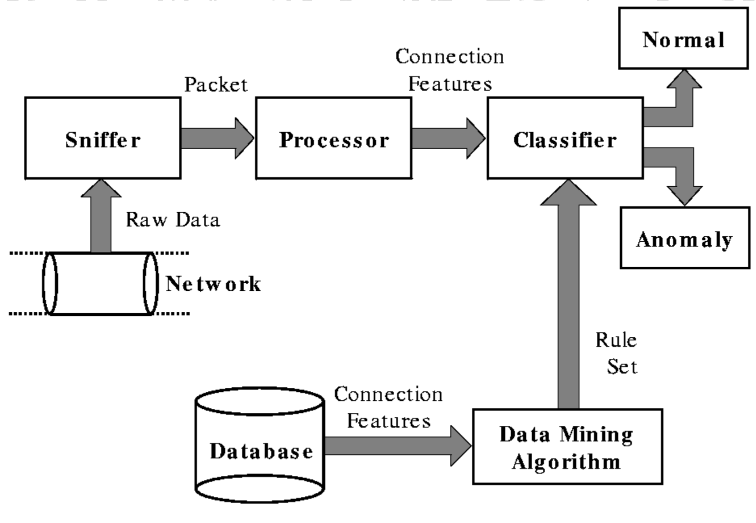


Figure 2.3: Reference Framework Model

In this section we present our framework for real-time intrusion detection using classification techniques. The overall model is composed of two parts: the former is related to the classification process, and consists in extracting behavioral models from pre-elaborated network traffic, building the knowledge starting from a database of labelled connection features; the latter is a real-time intrusion detection system which analyzes and classifies network traffic based on the models inferred (Figure 2.3 on the preceding page). In particular, we execute the off-line pattern recognition process on a data set in order to extract a set of rules or classification criteria; such a set is then used in a real-time classification process implemented by the IDS that analyzes these pre-computed classification criteria and compares them with informations evaluated by real-time network traffic. Each connection (as defined on page 34) can be associated to an array of features, and thefore represented in a vectorial space: the classifiers partition this space in a *normal* region and an *attack* region. For representation's sake, Figure 2.4, only shows the simple case of a two features space. It is worth pointing out that such a representation has only been selected for presentation purpose, and is not referred to a specific classifier type. In order to implement an efficient clas-

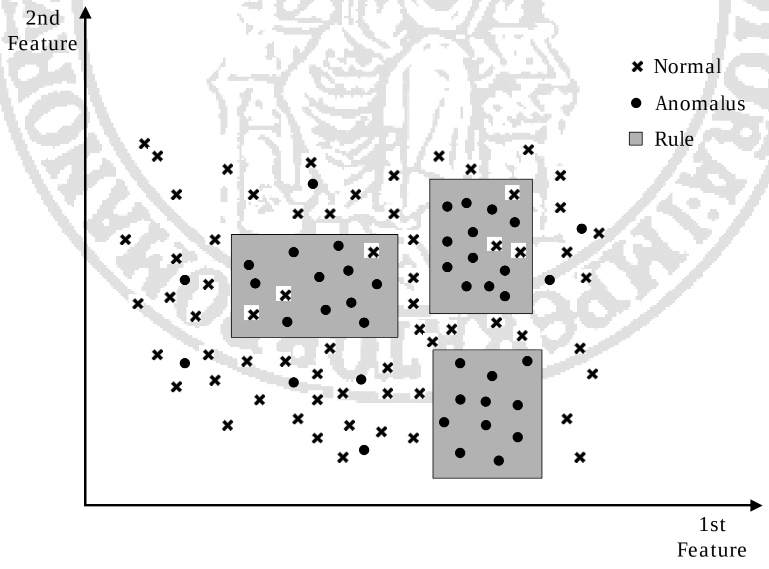


Figure 2.4: Vectorial representation in feature space

sifier, it is important to define a suitable set of features to be extracted from the network traffic contained in the database. The greater the capability of the set of features to discriminate among different categories, the better the classifier.

Defining Connection Features

There are three levels at which feature sets may be defined:

- The features may be referred to the single packet captured from the network:

  although this set is easy to compute, it is not able to detect all the potential attack types.

- A set of features related to the entire session which the packet belongs to may be defined:

  this is due to the fact that some intrusions may be realized by means of a sequence of packets belonging to either the same connection or different connections.

- The computed set of features may perform a statistical analysis of the relation between the current session and the other ones:

  this is needed in order to capture intrusions which affect the interrelation among different sessions.

To cope with the aforementioned requirements, we have adopted a model descending from the one proposed by Stolfo. We are interested in TCP, UDP and ICMP traffic. The features defined by Stolfo et al. can be classified in tree main groups: *intrinsic* features, *content* features, and *traffic* features. Intrinsic features specify general information on the current session, like the duration in seconds of the connection, the protocol type, the port number (i.e. the service), the number of bytes from the source to the destination, etc. (see Table 2.3 on the following page). The content features are related to the semantic content of connection payload: for example, they specify the number of failed login attempts, or the number of shell prompts (Table 2.4 on the next page). The traffic features can be divided in two groups: the *same*

duration	connection duration (s)
protocol_type	type of transport protocol
service	port number on the server side
src_bytes	bytes from source to destination
dst_bytes	bytes from destination to source
flag	status of the connection
land	land attack
wrong_fragment	number of wrong fragments
urgent	number of urgent packets

Table 2.3: Intrinsic Features

host and the *same service* features. The same host features examine all the connections in the last two seconds to the same destination host of the current connection, in particular the number of such connections, or the rate of connections that have a "SYN" error. Instead, the same service features examine all the connections in the last two seconds to the same destination service as the current one. These two feature sets are defined *time-based* traffic features because they analyze all the event occurred in a time interval of two seconds (Table 2.5 on the following page); some types of attacks, instead, such as slow probing, may occur every few minutes. Therefore these features could not be appropriate to detect all the attack types. To this aim a new set of traffic features, called *host-based*, has been defined; the same host and the same service traffic features are also computed on a window of one hundred

hot	number of hot indicators
failed_logins	number of failed login attempts
logged_in	successfully logged in
compromised	num compromised conditions
root_shell	root shell is obtained
su	su root command attempted
file_creations	number of file creations
shells	number of shell prompts
access_files	number of file accesses
outbound_cmds	outbound commands in ftp
hot_login	the login belongs to the hot list
guest_login	the login is a guest login

Table 2.4: Content Features

connections rather that on a time interval of two seconds. In our framework we will adopt only the intrinsic and the traffic features. Our purpose is to realize network-based intrusion detection system, whereas the content features are more adapted in a host-based scenario. Thanks to the access to the operating system's audit trails or system logs, an H-IDS is more efficient in the analysis of the dangerous commands execution on a single host.
As stated earlier, we used this predefined set of features, since if proved effective in detecting attacks within the dataset we used for experiments. Other features could also be defined, and easily implemented for real time detection with our preprocessor. Discussion about the issue of defining features will be discussed in section 2.3.1 on the next page, with reference to the problem of botnet detection.

Same Host	
count	number of connections to the same host
serror_rate	% of connections with SYN errors
rerror_rate	% of connections with REJ errors
same_srv_rate	% of connections to the same service
diff_srv_rate	% of connections to different services
Same Service	
srv_count	number of connections to the same service
srv_serror_rate	% of connections with SYN errors
srv_rerror_rate	% of connections with REJ errors
srv_diff_host_rate	% of connections to different services

Table 2.5: Time-Based Traffic Features

The proposed real-time IDS architecture consists of three components: a *sniffer*, a *processor*, and a *classifier*. The sniffer is the component at the lowermost layer of the architecture; connected directly to the network infrastructure, this module captures all the packet on the wire. The sniffing is possible by setting the network card in promiscuous mode. Usually the sniffer also decodes the packets, translating them in a human-readable format.
The processor component elaborates the packet captured from the sniffer in order to extract the selected set of features. The main issue of the features computation process is related to the need of keeping up-to-date informa-

tion about the current connection, as well as the other active connections. We have to keep in memory a representation of the current network status in order to evaluate the statistical relations among the active connections. Data in memory have to be properly organized in order to reduce the features computation time.

Defining Features for Botnet Detection

In this section we will show how, by analyzing another application context, it's possible to define feature representing a different model. From the reference model's standpoint, the problem will still remain the classification of an entity represented by a vector of features. We'll describe here the feature definition problem for botnet detection.

Botnets can be considered as distributed platforms for executing malicious actions. Though no comprehensive botnet taxonomy has been proposed yet, several have been presented so far [25, 11, 12]. One of the properties which easily separates botnet classes is the type of command and control channel used. Such a choice in fact, dramatically influences botnet effectiveness, and its resiliency to the defenses deployed by network and system administrators. The problem of detecting botnet characterized by a centralized command and control channel, such as those based on IRC or HTTP [88] with a classification theory approach [14] will be addressed here. Such a structure for the command and control channel grants a high degree of simplicity in implementing the control function, performed by the botmaster via the only available channel. Yet, it suffers from the drawbacks of the presence of a single point of failure. In fact, once the command and control channel is identified, the whole botnet is dismantled and torn down. That's the main reason why we address centralized botnet detection at the moment, and we try to detect such botnets as soon as possible, during their lifecycle. In the botnet model we decided to address, any infected host scans a defined subset of the whole IP space looking for known vulnerabilities. Once a vulnerable host is found, the vulnerability is exploited, and the host gets infected. Once infected, the new bot contacts a server where he downloads the bot binary and then joins the botnet as a fully operating soldier at the control of the botmaster.

DNS-Based Botnet Detection The first examples of bots used to have the botmaster and command and control IP hardcoded, hence once either of those was discovered, the botnet was made completely useless. For both flexibility and resiliency matters, though, bots started using symbolic names for contacting both the command and control channel and the botmaster. That is why a new bot, once infected, will issue a number of DNS queries to resolve the symbolic names associated to the entities it has to contact. By analyzing DNS requests, hence, we can hopefully intercept and eventually stop a bot before it connects to the rest of the botnet and gets involved in malicious actions. Furthermore, bot-issued DNS queries can allow to detect and tear down the whole command and control channel, and then the whole botnet, at once. For that reason, we propose to analyze the statistical properties of DNS queries [91]. In order to make such analysis effective, we have to imagine which may be the typical behavior of a bot once it's been recruited or, if it was sleeping, once it awakes back, and after that, when it tries to connect to the command and control channel. Experiments show that known bots are characterized by a propagation profile similar to that of popular internet worms. That is because many bots inherited their own propagation strategies from some popular worms of the past, which proved very quick and effective [87, 105]. Under this assumption, we can assume that the number of infected hosts, and hence of DNS requests issued, varies according to a sigmoidal law. At the beginning, very few hosts are infected. Few hosts, then, perform scans, and the probability of clean hosts to get infected is really low. Once the number of infected hosts increases, they are spread all across the IP space. In this phase, the propagation speed is very high, and increases very quickly. Beyond a certain number of infected hosts, instead, the propagation speed decreases. This happens because the probability of finding a non infected host during scans decreases, and also due to some administrators' reaction. Once the vulnerability and its exploitation are discovered, in fact, some of the bots will be sanitized and removed from the botnet. Hence, it's difficult to identify properties which can allow to detect a botnet during the initial and the final phases of its life. By observing DNS traffic properties during the phase associated to the steepest part of the sigmoid, it is possible to imagine features which can allow to effectively discriminate between legitimate and botnet-related requests. The definition of such features, of course, depends on the application context and on the problem we are willing to address.

Language-based Botnet Detection Once the botnet is established and in a steady state as to its spreading, it becomes more difficult to detect its activity from the network activity point of view. In fact, its related statistics will look steady and hardly distinguishable from background traffic characteristics. Hence, it might make sense to think about some detection techniques based on packet inspection, which exploit information at the application layer. Of course, in order to be able to do that effectively, we have to consider all the well known drawbacks related to packet inspection, which has been criticized lately. In general, application level payload decoding is not feasible in high speed networks, since it's very hard to keep up with network traffic's pace with a detailed analysis. Hence, we need to filter out the largest portion of traffic not of any interest for our botnet hunting purpose. If we decide to analyze botnets characterized by a centralized command and control channel based on either IRC or HTTP protocol, for example, we can exclude from our analysis any traffic flow not using either of such protocols. Of course port based protocol identification is the most straightforward method to do that, but it is completely ineffective when port numbers are used, other than the standard ones. Many techniques have been proposed and implemented to cope with blind protocol identification, and some of those are also embedded in common, production level Intrusion Detection Systems, such as the PIA module for Bro [46]. Once the correct application level protocol is selected and isolated from the rest of the traffic, application level payload inspection becomes feasible again, and it might even be effective on high speed links. Initially, we want to study the case of IRC based botnets. In such botnets, bots subscribe to certain IRC channels, and commands are usually issued by the botmaster via the channel topic or some either broadcast or private messages. IRC channels are typically used by humans to intercommunicate. Hence, we think it might be possible, by analyzing IRC channel logs, to discriminate between normal IRC channels, and botnet-related channels. In fact, we expect to be able, by using natural language recognition techniques, to distinguish between humans having a conversation and bots responding to commands issued by the botmaster. Our intuition is that a bot, due to its limited set of commands, will have a limited dictionary, resulting in a limited number of used terms, and a low variability of each sentence's properties. Since bot commands are structured as common shell commands, we expect to find a very structured set of sentences in a botnet channel. To be more specific, we expect most of the sentences of a bot-related conversation to look like a command-arguments-values sequence. On the other hand, a human conversation should be characterized by a higher variability of sentence properties, a different interaction pattern among chatters, and possibly a broader dictionary. We aim at defining some features, also borrowed from natural language analysis theory, which allow us to detect bot channels. By analyzing them by means of pattern recognition techniques, we can implement anomaly detection easily, since "clean" IRC channels are easily available for training.

2.4 Real-Time IDS Implementation Issues

The reference model depicted in figure 2.3 on page 35 has been discussed so far only with respect to the classification aspect of the network security issue. We also implemented the proposed architecture in practice, and in this sec-

tion we'll show how it can be employed to realize a real-time network based intrusion detection system, by monitoring the network traffic in order to extract a set of features from it, as well as performing behavior classification based on the extracted features. Monitoring, in particular, is the most challenging issue to face from the point of view of a real-time analysis. In our architecture, the monitoring system can be divided into two components: the sniffer that captures traffic from the network, and the processor that computes both the *intrinsic* and the *traffic* features. While in an off-line analysis features computation is simpler, since all the information about connections are stored in a database, in a real time analysis statistic measures have to be be computed and the system status updated every time a new packet is captured from network[46].
In order to extract features from the traffic, an effective processor must ensure two requirements:

- holding information about the state of the connection which the analyzed packet belongs to;
- holding comprehensive information about the traffic flows that already have been seen across the network.

According to the definition proposed in the previous section, every packet can be considered as a single unit that is inserted in a more complex structure, namely the *connection*, and about which the features are computed. While neither UDP nor ICMP traffic requires a heavy load of computation, TCP traffic requires to emulate the TCP state diagram both on the client and the server sides and for every active connection. In particular, when a new packet is captured, the system retrieves information about the connection such a packet belongs to, and updates the connection state of both the client and the server based on the TCP protocol specifications.
In order to compute the statistical relations, information about the past TCP, UDP and ICMP flows is required, including those connections which have been closed according to protocol specifications. Traffic features, in fact, are

computed by analyzing all the connections (either active or expired) having similar characteristics — besides the destination IP address and/or the destination port — to the current one. Every connection has to be kept in memory until it is not needed anymore for other computations.
Our architecture is implemented by modifying the open-source N-IDS *Snort*; we have used this system as the base framework on top of which we have built our components. Snort is a lightweight network IDS created by Marty Roesch. Its architecture is made up of four main blocks: a *sniffer*, a *preprocessor engine* that realizes a pre-computation of captured packets, a *rules-based detection engine*, and a set of *user output tools*. Thanks to Snort's modular design approach, it is possible to add new functionality to the system by means of *program plugins*. Moreover, Snort provides an efficient preprocessor plugin that reassembles TCP streams and can thus be used to recover the TCP connections status.

2.4.1 Calculating Traffic Features

We have implemented a new preprocessor plugin which computes the connection features. The main issue we tackled has been the computation of the traffic features, which requires that a proper logical organization of the data is put into place in order to recover information on the past network traffic. Moreover, to assure that the real-time requirement of the system is met, a fast access to stored data is mandatory.
As to the data structures, we have adopted a binary search tree. In the worst case this structure guarantees a performance comparable with a linked list, from the point of view of search time consumption; performance further improves in case the tree is a static and well-balanced one. Unfortunately, our structure is not a static tree because the connections are not known in advance; though, a self-adjusting binary tree can be adopted in this case in order to balance a dynamic tree.
We have used a Snort library of functions to manage the so-called *Splay*

Trees. A Splay Tree is an elegant self-organizing data structure created by Sleator and Tarjan [100]: it actually is an ordered binary tree, in which an item is moved closer to the entry point — i. e. the tree root — whenever it is accessed, by means of a rotation of the item with the parent node. This makes it faster to access the most frequently used elements than the least frequently used ones, without sacrificing the efficiency of operations such as insert and search.
With the above mentioned tree structure, we have implemented two trees, a *Same Host Tree* and a *Same Service Tree* to compute the same host and the same service traffic features, respectively. Every node in the tree is identified by the destination IP address in the first tree, or by the destination service in the second one. In this way, we want to store in the same node information about all the connections that share the same characteristics. In order to compute both the time-based and the host-based traffic features, for every node in the tree we have implemented two linked lists, one for each set. The linked lists contain information like source IP address and/or source port for all the connections that have been identified and that have the same destination IP address and/or the same destination service (see Figure 2.5 on the next page). The elements of the list, one for every connection, are ordered in time: the first element is the oldest one, the last is the most recent. When a new packet is captured from the network, our preprocessor plugin first analyzes the protocol of the packet in order to identify the most appropriate procedure to compute the intrinsic features. If the packet belongs to either a UDP or an ICMP traffic, the information required to compute the intrinsic features is entirely contained in the packet. In case of TCP traffic, the procedure recovers the session which the packet belongs to in order to determine some crucial information, like the duration of the connection or the number of bytes sent along both directions of the stream, that cannot be directly inferred from the packet. Then, the procedure analyzes the destination IP address and the destination port to compute the traffic features. The searches in the two trees are performed: if no node has been found, a

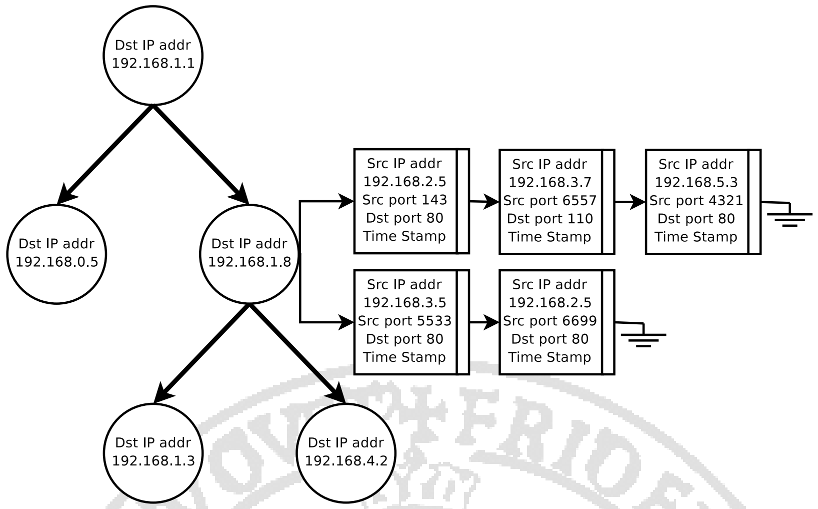

Dst IP addr 192.168.1.1
Dst IP addr 192.168.0.5
Dst IP addr 192.168.1.8
Dst IP addr 192.168.1.3
Dst IP addr 192.168.4.2
Src IP addr 192.168.2.5 Src port 143 Dst port 80 Time Stamp
Src IP addr 192.168.3.7 Src port 6557 Dst port 110 Time Stamp
Src IP addr 192.168.5.3 Src port 4321 Dst port 80 Time Stamp
Src IP addr 192.168.3.5 Src port 5533 Dst port 80 Time Stamp
Src IP addr 192.168.2.5 Src port 6699 Dst port 80 Time Stamp

Figure 2.5: Same-Host Tree Structure

new one is created, and the traffic features relative to the current connection are set to zero. If a node is already in the tree, the procedure analyzes the two linked lists to compute the statistics for both time-based and host-based traffic features. Every element in the list is analyzed and the statistics are updated. During this process the elements that do not belong neither to a time interval of two seconds, nor to a window of the latest one hundred connections are pruned.

2.4.2 Testing the Approach

In this section we evaluate the overhead on the performance of the whole system caused by the operation of the IDS, pointing out the increase in CPU usage and memory consumption with respect to the values observed while running Snort without our plugins. Our purpose is to show the affordability of real-time intrusion detection, using techniques which are usually employed in off-line analysis. We evaluate both CPU and memory overhead, and packet loss ratio. Such tests are deployed in two scenarios: in the first case, we build a testbed to emulate network traffic in a controlled environment; in the second, we sniff traffic flowing on the local network at Genova National Research Council (CNR). In this scenario, the most important re-

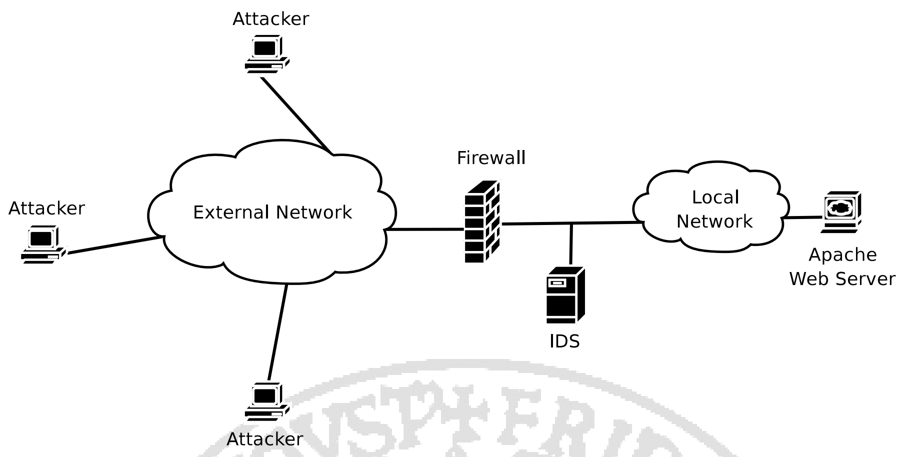


Figure 2.6: Testbed for Real Time IDS

sults are those regarding packet loss, as we want to show that the further analysis steps needed to carry out detection using such techniques don't affect dramatically the percentage of lost packets, with respect to the total sniffed traffic, thus demonstrating the affordability of intrusion detection with such techniques. While working on the testbed, we consider the topology depicted in Figure 2.6.

In order to work in a totally controlled environment, we have to emulate the depicted scenario rather than working in a real network environment; for that purpose, we use another topology which just emulates the one depicted above, as drawn in Figure 2.7 on the next page. Furthermore, we test the IDS using it on a real and heavily loaded network, whose topology is drawn in Figure 2.8 on page 50. Such a test is useful to understand some of the limits of our plugin, and to individuate some development directions. Indications help us understand the drawbacks of our implementation choices. In table Table 2.6 on the next page we see the values of CPU overhead due to use of Snort alone, version 2.1.0, and Snort plus our plugins. The machine operating as IDS in the emulated traffic scenario is equipped with a 1GHz Pentium III CPU and an amount of 256MB RAM, running Mandrake Linux 9.1 as operating system, kernel version 2.4.19. In this case we can point out a very low, almost inexistent increase in memory consumption (Table 2.7). The

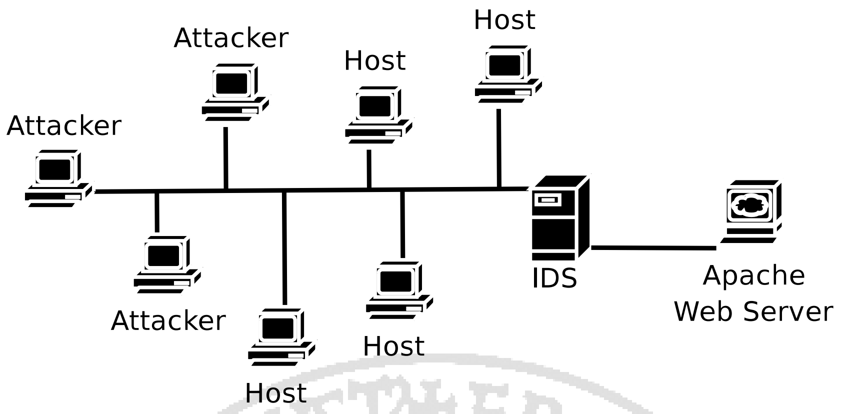


Figure 2.7: Testbed for Traffic Emulation

doubling in CPU usage percent, when using the modified version of Snort with respect to case of Snort alone, is not such a negative result, since overall CPU usage is still low and under reasonable thresholds, also considering that we are using general purpose, not dedicated, hardware. The extensive

	Snort-2.1.0	**Snort + Plugins**
Emulated Traffic	0.12%	0.22%
CNR Traffic	1.16%	2.42%

Table 2.6: Average CPU Overhead

test on CNR network still shows a slightly higher CPU usage for the modified version of Snort, still within the limit of 8% overall overhead. The machine acting as IDS is equipped with a 2GHz Pentium IV, 512MB RAM and RedHat Linux 8.0, using kernel 2.4.18. Once again it is worth pointing out that

	Snort-2.1.0	**Snort + Plugins**
Emulated Traffic	1.69%	1.70%
CNR Traffic	4.99%	9.46%

Table 2.7: Memory Overhead

the results of our measures must be looked at considering that the hardware we use is not dedicated to intrusion detection, indeed we use general purpose personal computers with a suitable software instrument installed and run-

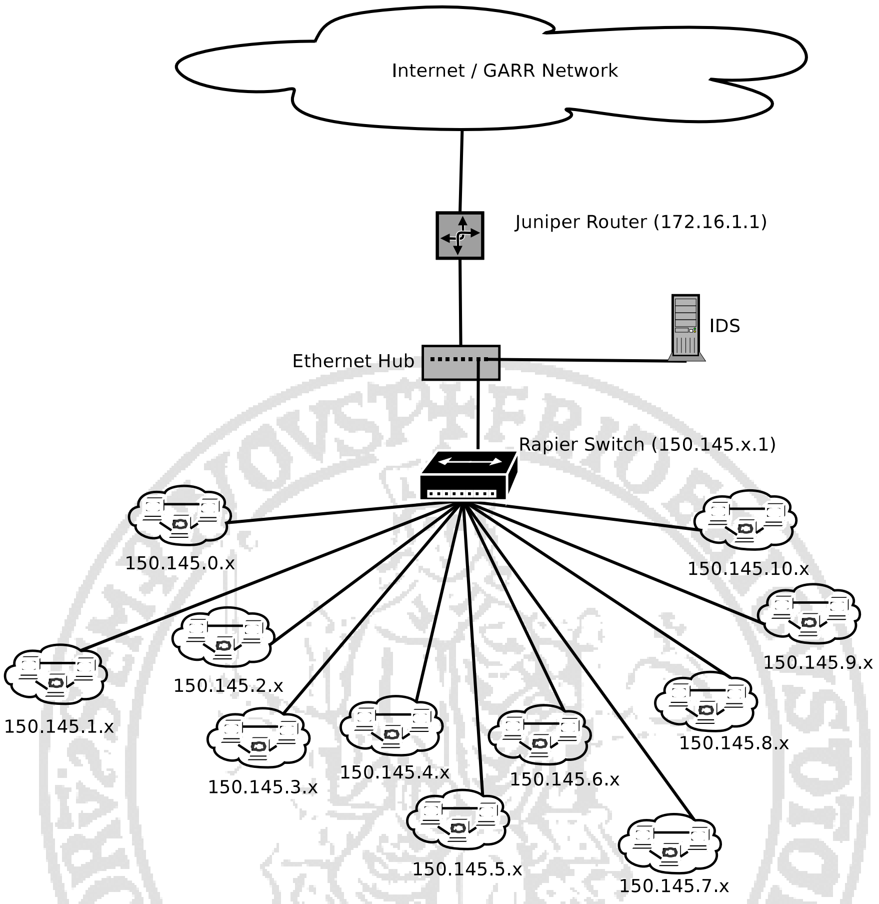


Figure 2.8: CNR Network Topology

ning.

Of course, the most interesting indication regards the packet loss ratio. To attain the best results in intrusion detection, the main requirement is not to lose any packets, no matter how much of the system resources we use, if affordable with the available hardware. Such result is sketched in Table 2.8 on the following page. In the test deployed using emulated traffic, we notice an increase of less than 10% in packet loss with respect to the clean version of Snort, though the values are lower than the ones obtained testing the system on a real network. This may be ascribed to the hardware used in the two cases: the setup used in the latter scenario is much better than the one used in the former case. In both cases, anyway, we observe a very low increase in

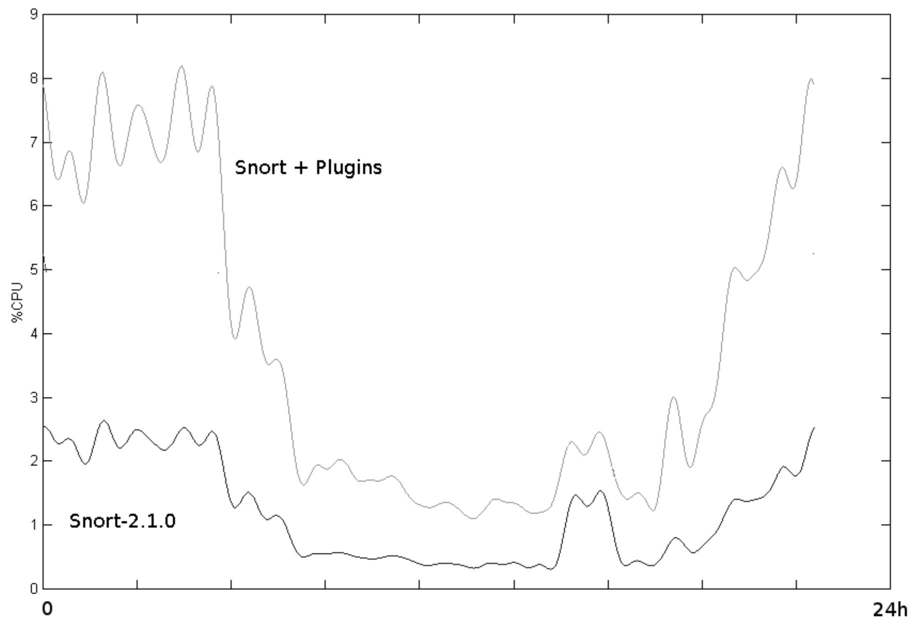


Figure 2.9: CPU Usage - CNR Network

packet loss ratio, showing the affordability of such a technique.

	Snort-2.1.0	**Snort + Plugins**
Emulated Traffic	0.39%	0.42%
CNR Traffic	0.14%	0.16%

Table 2.8: Packet Loss

2.4.3 Key Findings

This section shows that it is possible to combine real-time intrusion detection with data mining techniques, keeping the system overhead under reasonable thresholds and containing the packet loss ratio within certain boundaries. The system was tested on a particular definition of features. Obviously, support is provided for the easy implementation of different types of features. This makes the system a flexible tool for calculating selected models to describe network traffic. As a matter of fact, the only input the system accepts is raw network traffic, and no assumption is made about it. As long as a proper preprocessing function is implemented, virtually every type of feature can be computed about the traffic. The system has been projected with the idea of using it with network scenarios of limited size. Obviously bigger networks might require architecture using multiple real-time modules, such as the one presented in 4.3. By using several monitors spread over the network, it's possible to reduce the load of each of them, by dividing the total load over multiple monitoring systems. The issue of coordinating information coming from multiple sources can be dealt with by using solutions such as the ones proposed in chapter 4 on page 95. For this purpose, in fact, the computed features and the classification results, can either be stored for further analysis, or sent over the internet to another hosts which may run a suitable software for collecting and interpreting such information coming from multiple sources.

Chapter 3

Labelling Network Traffic – Data collection and related issues

In previous chapters we introduced the employment of pattern recognition techniques for the problem of network security. More in details, we propose to use such techniques for traffic classification, in order to discover malicious activities. As described in section 2.1.2 on page 27, both the supervised and unsupervised paradigm for classification can be useful for application in network security. Also, we aim at using both misuse based and anomaly based classifiers based on pattern recognition techniques, to overcome the well known problems of signature detection techniques, which can be evaded by means of simple attack modifications. Several examples of such classifiers can be found in literature, for example in [43]. Some of them need to be trained in a *supervised* fashion. The main advantage of a pattern recognition approach is the ability to generalize. This property can be very useful when aiming at detecting novel attacks without the need for a complete description of all the possible attack signatures. In [61, 118] some other classifiers are used, which can be ascribed to the more general category of *unsupervised* systems based on the novelty detection approach [98], i.e. the identification of new or unknown data or signal that a system is not aware of during training. Anyway, it is worth noting that an IDS realized by means of a pat-

tern recognition approach based on supervised learning techniques typically performs better than those based on unsupervised learning techniques [62]. Obviously, as suggested by the name, supervised systems need a *supervisor* during their training. Usually, that role is performed with the intervention of a human expert of the problem at hand. In practice, the supervision is performed by feeding the system, during its training phase, with a dataset of exactly labeled entities. In the case of network traffic classification, a dataset is needed made of packets, each associated to a label describing the class it belongs to. In our case, we won't aim at detecting if the specific class of attack a packet belongs to, if any attack is detected. Yet, we want to detect whether an anomalous malicious activity is in progress. Therefore, we will only consider two possible categories for packets, namely *Normal* and *Attack*.
Due to the previous considerations, in order to build a system which is effective at detecting intrusions in a particular network by also using supervised classifiers, we need to have a suitable dataset of labelled packets available. The dataset needs to be well representative of the problem at hand, it must contain a good sample of the possible behaviors which will be found on the real network, and for that reason it doesn't have to be outdated. Also, it needs to contain packets whose labelling is reliable enough, not to induce the classifiers to commit unwanted error, due to incorrect biasing.
Unfortunately, one of the main drawbacks occurring when using PR-based system in real environments is the difficulty in collecting a representative labeled set of data for training them. This is due to several factors, among which the difficulty in correctly labelling, by hand or at least with the supervision of human experts, large amounts of data. Also, network traffic usually carries huge amounts of private information. Hence, few people are actually willing to distribute their own dataset, thus generating a big problem within the scientific community doing research in network security. In fact, it is very difficult to ensure the reproducibility of scientific experiments in that field, since there is no dataset which is commonly acknowledged and widely used within the community. This aspect is often disregarded in papers propos-

ing pattern recognition approaches for coping with the intrusion detection problem, where usually standard labeled databases, such as the well-known *KDD 99* [32], are used for testing [42] [68] [119]. This choice is no longer accepted in the IDS community, since network traffic has now significantly changed and some papers (see for example [75, 72]) highlighted many flaws in the construction of such database. There are obviously some advantages and disadvantages in using traffic traces created synthetically. The two possible approach are network simulation and network emulation. By using the former approach, it is possible to simulate a network scenario created ad-hoc in order to suit the problem at hand. The traffic can be completely controlled, and attacks have to be injected on purpose by who runs the simulation. Both attack and normal traffic characteristics have to be configured by the operator, who has to have a deep knowledge of both the network scenario and the problem at hand.

A possible solution [28] could be the use of *unsupervised* anomaly detection techniques, as the one proposed in [33]. Here, the authors suggest using an unsupervised algorithm for recovering the anomalous elements from an unlabeled set of data. After anomalies or intrusions are detected and removed, it could then possible to train a misuse detection algorithm over the polished data. In the above cited papers, however, the problem of verifying to what extent such data can be used for effectively training a misuse-based approach is not addressed at all. Also, the employment of a single technique might produce biased results, since a single classifier might be good at detecting specific attacks, whereas it might be bad at detecting some other.

3.1 Privacy Issues in Network Traffic Collection

As stated before, prior to discussing methods and systems capable of generating widely distributed, reliably labelled traffic traces within the research and industrial community, it is worth considering some serious problems related to privacy issues in spreading the data. Due to regulations common to most of the countries, in fact, traffic archives can't be freely distributed not to break privacy related laws. To such extent, it's useful to have tools which eliminate sensible information from network traffic, making traffic traces widely available.

3.1.1 Network Traffic Anonymization

In order to cope with the problem of network traffic anonymization, we developed a tool which is flexible, easy to use, and multiplatform. Even though several such systems are available, we developed our own with performance in mind. We'll describe the used technique and show that its performance is comparable with that of well known anonymization tools. Also, by introducing additional options with respect to other well-known anonymization tools, we will show how it is still possible to keep the resource usage under reasonable limits. Several approaches have been proposed for network traffic anonymization. Some propose methods aimed at avoiding the distribution of the traces, totally preventing information leakage [78]. This can be very useful in some particular applications, but is completely against our purpose of collecting traffic traces with the aim of sharing them. In fact, our primary goal is to create the traces in order to simulate and evaluate the performance of security systems, such as Intrusion Detection Systems (IDS). In this field, traffic traces are typically used to estimate and eventually validate models of either normal and anomalous traffic for a particular network environment. In such cases a different approach must hence be used. The availability of un-anonymized traffic archives might allow a malicious user to reconstruct the activities of each user on the network, harvest passwords, bank account and credit card numbers, thus exposing the users to many risks. Also, some administrators might be interested in keeping the communication patterns among hosts on their network secret. This, for example, can be useful to hide the internal structure of a natted network, in order to disguise the real private IP address of a very important server, to prevent malicious user to

target sensible machine with specific and disruptive attacks. Also, before distributing network traces, an administrator might be interested in deleting information about the type of hardware used, in order to make machine identification more difficult, or eventually impossible from the hardware point of view. These consideration evidentiate the need for a layered anonymization. According to such different requirements, anonymization has to be carefully projected at each layer were we want information to be kept private. The most simple and obvious of those ironically consists in barely deleting all the sensible data, and replacing it with any random symbol. Not only this keeps all the private information safe, but also permanently deletes any other information which can be useful in research. What we want to do, instead, is to keep the information about mutual relationships among hosts, thus making the details of their communication unrecoverable. In particular, we implemented the possibility of anonymizing header fields at Link State, Network and Transport Layer. Furthermore, our tool also deletes sensible information at the Application layer, by obfuscating the whole payload, though keeping the packet size intact. To such purpose, we substitute the whole payload content with meaningless random bytes, and reevaluate the packet checksum in order to obtain a traffic trace containing valid packets. What other tools usually do, in fact, is truncating the payload, thus obtaining malformed packets which are not always analyzable by means of traffic sniffers, and also altering the distribution of amounts of data exchanged on the network.
The type of anonymization strategy adopted is tightly related to the application at hand. In Sections 3.2 on the next page and 3.3 on page 59 we will describe the expected application context for our tool and, after defining the requirements, we'll introduce the techniques used to anonymize each packet accordingly. Finally, in Section 3.3.4 on page 63 we will show the tool's functionalities, and we'll compare its performance with those of two other well-known anonymization softwares [1, 52].

3.2 Anonymization in Practice

In this section we will introduce some of the observations that led us to our implementation of an anonymizer. As stated before, the problem will be deal with in a layered fashion. Due to the layered structure of network protocols, in fact, the issues of each layer should be treated separately, in order to keep the required crosslayer independence and transparency.

3.2.1 Header Anonymization

In order for two applications to exchange data correctly, and for the packets to be well-formed, their header fields must be correctly formatted. In the context of privacy enforcement, we want to avoid the possibility of reconstructing any activity performed by each of the hosts and their users. We can choose to hide the information about the type of hardware used, the position in the network, the subnet a host belongs to, and also about the type of service the communication is bound to. According to each of the aforementioned requirements, an anonymization tool has to be able to modify respectively: the MAC address, the IP address and the ports. As to MAC addresses anonymization, as well as port numbers, the process is very simple: we only need to define an injective function for the transformation, which maps each value randomly into another. The only thing to care about is, obviously, to grant that equal values will remain such, even after anonymization. For IP addresses, instead, by using our approach, it is possible to either decide to use a random injective function with no constraints, or some other approaches aimed at saving some information which might be useful. We implemented both a class preserving and a prefix preserving transformation, as it will be shown in section 3.3 on the following page. As names suggest, the former aims at preserving the class an IP address belong to, also in the anonymized address domain, keeping the class an address belong to consistend through the transformation. The latter instead aims at keeping consistent the longest common prefix of all the anonymized IP addresses. Thus, two addresses sharing, let's say, the first 12 bits, will have the same 12 bits in common also in the anonymized address space, even though they might have a different value.

3.2.2 Payload Anonymization

While the header contains information about the sending and receiving hosts of each packet, the payload contains data produced by the applications communicating through the network. Hence, that data is directly associated to what the users are doing, and the semantics of the data they are exchanging. Usually private data are exchanged, as well as unencrypted passwords and bank account and credit card numbers. Also, private conversations dealing delicate matters might occur. The simplest approach to anonymization consists in completely deleting such information. Yet, the payload can either be substituted by random symbols or abruptly truncated. In the latter case, malformed packets are generated, as the size field in the header won't correspond to the actual size of the packet, whereas in the former the packet checksum will have to be recomputed, as its value is related to the actual content of the payload. Our tool implemented the former strategy, by using an incremental technique for recomputing the checksum described in paragraph 3.3.3 on page 63.

3.3 Anonymizer Implementation Details

In this paragraph we will discuss some of the implementation details, by motivating the choices we made while developing our anonymization tool. We will present two strategies for anonymizing IP addresses, and some issues related to payload anonymization.
The developed software is available on Sourceforge[1] for download.

[1]http://anonymizer.sourceforge.net/

Class	**initial bits**	**net bits**	**host bits**	**starts**	**ends**
A	0	7	24	1.0.0.0	127.255.255.255
B	10	14	16	128.0.0.0	191.255.255.255
C	110	21	8	192.0.0.0	223.255.255.255
D	1110	-	28	224.0.0.0	247.255.255.255
E	1111	-	28	248.0.0.0	255.255.255.255

Table 3.1: IP Address Classes

3.3.1 IP Anonymization

Before illustrating in deep details how IP anonymization is performed, let's recall their structure. IP addresses uniquely identify an interface connected to a specific network. They are strings of 32 bits, usually divided into 4 groups of 8 bits (octets). For the sake of representation, IP's are usually written as four numbers ranging from 0 to 255 separated by dots (dotted decimal notation). Each address has a variable length prefix, identifying the subnet a host belongs to (network prefix), and a suffix, identifying the specific interface within the subnet. IP addresses have fixed length, but the network and host fields can have variable length, the only constraint being that both lengths sum up to 32 bits, obviously. Several fixed length for both the fields are specified, depending on the class an IP address belongs to. The original addressing policy used to consider four different classes, and a further fifth class reserved for future use. Such classes are usually named after the first five capital letters of the roman alphabet, namely from A to E: It worth pointing out that not all the reported IP addresses are available. Class D addresses are not actual host addresses, since they are intended for use with multicast, while class E addresses are reserved for future or experimental use. Also, among classes A B and C some IP addresses are reserved for special use. Some addresses are available only for local networks, and not routed on the internet, such as:

- **Class A:** 10.0.0.0 - 10.255.255.255
- **Class B:** 172.16.0.0 - 172.31.255.255

- **Class C:** 192.168.0.0 - 192.168.255.255

Several solution exist, which are limited by the trade-off between the security of the anonymization scheme, and its effectiveness. A first solution is represented by the so called *black marker* [99] anonymization. According to such a scheme, all IP addresses are substituted by a constant IP. Obviously, using this method results in an unrecoverable loss of information. Alternatively, it is possible to use the *random permutation* technique. In this case, a random anonymized address is uniquely associated to each IP address. A third method is based on truncation. In this case, a number of bits to preserve is chosen, and all the remaining bites will be set to 0. In this case, as for the black marker technique, a large number of addresses will be anonymized in the same way. Only subnetting can be preserved. As an example, in table 3.3.1 we report some IP addresses, and their anonymized versions. The letter c indicates a generic constant.

IP	**Blackmarker**	**Random Permutation**	**Truncation**	**Class Preserving**	**Prefix Preserving**
141.142.96.167	c	12.72.8.5	141.142.0.0	127.0.0.1	12.131.12.67
141.143.96.18	c	12.161.3.3	141.143.0.0	127.0.1.1	12.55.79.102
141.142.132.37	c	212.3.4.1	141.142.0.0	127.0.0.2	12.131.201.29
12.161.3.3	c	12.72.8.5	12.161.0.0	1.0.0.1	187.192.32.51
12.72.8.5	c	141.142.132.37	12.72.0.0	1.0.0.2	187.78.201.97
212.3.4.1	c	141.143.96.18	141.142.0.0	192.0.0.1	31.197.3.82

Table 3.2: IP anonymization using different techniques

Class Preserving

Class Preserving anonymization is a strategy mainly developed for IP address anonymization. Its aim is to implement an injective transformation between original and anonymized /8, /16 and /24 class subnets. This means that, for example, if two addresses belong to the same original /16 subnet, they will still belong to an anonymized /16 subnet which is the same for both, even after transformation. Furthermore, the class of IP addresses will be preserved: class A, B and C IP addresses will still belong to the same class after anonymization. Hence, the first bits of the addresses will be preserved unchanged in order to maintain the address class. Also, private and public address classes will be preserved. Anonymized addresses will be chosen sequentially, starting from a common seed. This still ensures the privacy of anonymized information, since the original IP address is unrecoverable.

Prefix Preserving

This anonymization strategy allows to keep IP addresses grouped according to the longest prefix they have in common. In other words, if two IP addresses share an M bits long common prefix, they will still share an M bits long common prefix after anonymization. This automatically allows us to transform IP's belonging to the same subnet into IP's belonging to the same anonymized subnet. More formally, we use the following definition [1]:

two n bits long IP addresses $a = a_1 a_2 \ldots a_n$ and $b = b_1 b_2 \ldots b_n$ have a k bits common prefix, $0 \leq k < n$ if and only if $a_1 \ldots a_k = b_1 \ldots b_k$ and $a_{k+1} \neq b_{k+1}$.

3.3.2 *Signature* Preserving Anonymization

Since we want our anonymization tool to be used as a support to research on network security problems, we want to allow transformed traffic traces to be used for the evaluation of signature-based IDS, too. Hence, given a database of known signatures (as, for example, the ones used by *Snort*[2]), we enable our software to obfuscate all the payload but the desired signature. Then, in case a predefined string is found within the payload, it is not overwritten, but

[2]http://www.snort.org/

simply rewritten in the same way and at the same position in the anonymized payload as well.

3.3.3 Checksum Correction

As stated before, we want to allow our anonymizer to be able to keep the packet size unchanged, by obfuscating the information contained in the payload, instead of deleting it. In such a case, though, it's necessary to recalculate the checksum for each packet. In order to keep the anonymization time for each trace as low as possible, we decided to implement a strategy for incremental checksum update, described in [10, 73, 92], and generally used at the router side when the TTL value is changed in the header field of the packet. Let's define

HC	Old checksum
C	1's Complement of the old header sum
HC'	Updated checksum
C'	1's Complement of the updated header sum
m	Old value of each 16 bit field
m'	Updated value of each 16 bit field

For the header: $C' = C + (-m) + m'$. Then, for the whole packet checksum: $HC' = \sim (C + (-m) + m') = \sim (\sim HC + \sim m + m')$

3.3.4 Experiments on Anonymization

In order to prove the effectiveness and usability of our tool, we compared it with other two well-known tools, namely `tcpdpriv` [1] and *AnonTool* [52]. To such aim, a brief resume of the functionalities of these tools is presented in table 1, while in table 2 the execution time of each anonymization tool on different traffic traces is shown.

	Class Preserving	**Prefix Preserving**	**Payload Darkening**	**Checksum Correction**	**Port Anonymization**	**Live Anonymization**	**MAC Addr. Anon.**	**Pattern Matching**	**Linux Compatible**	**FreeBSD Compatible**
`tcpdpriv` [1]	●	●		●	●				●	
AnonTool [52]		●	●	●	●	●		●		
Anonymizer	●	●	●	●	●	●	●	●	●	●

Table 3.3: Functionalities of different anonymization tools

(a) Linux - *prefix preserving* strategy

	0.5 GB	**1 GB**
Anonymizer	$46s$	$1m46s$
AnonTool	$46s$	$1m43s$

(b) FreeBSD - *class preserving* strategy

	0.5 GB	**1 GB**
Anonymizer	$26s$	$1m53s$
`tcpdpriv`	$26s$	$1m52s$

Table 3.4: Anonymization time using different strategies and operating systems

By considering the data reported in both tables, it is possible to conclude that we were actually able to develop a tool that extends the functionalities provided by [1] and [52], by keeping execution times practically unchanged.

3.4 The Dempster Shafer Theory for Classification

In this section we propose a general methodology for automatically obtaining a dataset of labeled packets starting from raw tcpdump traces. Such dataset can be then used during the training phase of supervised pattern recognition algorithms. In particular, we present an architecture for automatically building up a network traffic database made up of packets, each labeled as either *Normal* or *Attack*. To build such architecture, we propose to use multiple classification techniques, therefore it is necessary to employ some information fusion strategy in order to combine them. Since we have no prior knowledge about traffic characteristics, what we need is an information fusion technique which neither relies on prior knowledge about the problem, nor on any configuration chosen according to the problem's nature. What we need is a rule which completely disregards the choice of the base classifiers, and doesn't change with the application domain. One such technique is represented by the Dempster-Shafer [44] theory, which completely disregards any prior knowledge about the data to be classified. The Dempster-Shafer combination rule, in fact, was defined once and for all, and completely removes assumptions about the base classifiers. Note that, though some works already propose to exploit the results of such a theory for intrusion detection (see for example [19, 117]), its use for labeling raw network traffic is novel. We will also evaluate to which extent a dataset labeled by using our approach can effectively be used to train a supervised classifier based on pattern recognition techniques. In order to do that, we've tested the system on a very well known database, made up of correctly labelled packets. We trained the chosen test-classifier by using both the real labels of the traffic and those calculated by means of our proposed system, and compared the obtained results. Moreover, we will show how the system is capable of improving its *knowledge* about analyzed traffic. By adding, in successive steps, more classifiers to the defined architecture, we show how it is possible to automatically improve the classification results. Moreover, we show how such a classifier can be a supervised classifier, trained by means of the previously known information. The system is able to pass its knowledge along to some supervised classifier and, since its training is complete, it can contribute with additional knowledge to the accuracy of the labelling process. The knowledge increase process could continue indefinitely, in an iterative fashion. It's not reasonable, though, to imagine that any classification system can perform beyond certain limits. For that purpose, in order to avoid useless overtraining, or efforts which are not justified by the obtained results, we've also studied a strategy to evaluate the relative variation in performance, in order to asses a sort of termination condition for the labelling process. Obviously, such con-

dition is related to the application scenario, and to the results expected by the operators who use the system for automated packet labelling.
It is worth noting that the choice of the Dempster-Shafer theory allows us to define a suitable *index* that is used for express the reliability of the packet labeling. In this way, unreliably labeled packets can be discarded from the database, and eventually saved for further processing. We evaluate how this choice can affect the training and generalization performance of the considered IDS as well as the quality of the labeling itself. The final aim of this study is to obtain a system which, in a completely automated fashion, is able to produce a dataset of labelled traffic, starting from raw traffic traces. Due to the iterative nature of the described process, it's not at all meant for either online use, or for real time attack detection. The process might in fact converge after several iterations, which renders its real time usage unfeasible.
The theory of Dempster and Shafer (D-S theory) has been frequently applied to deal with uncertainty management and incomplete reasoning. It is worth noting that it is different from the classical Bayesian theory, since for Bayes the sum of $P(A)$ and $P(\neg(A))$ always equals one, while this is not necessarily true according to the D-S theory. In fact, differently from the classical Bayesian theory, D-S theory can explicitly model the absence of information, while in case of absence of information a Bayesian approach attributes the same probability to all the possible events. The above described characteristics can be very attractive for managing the uncertainties of the network security domain, due, for example, to the presence of the zero-day attacks. There is also another main difference between the Bayesian theory of probability and the Dempster Shafer theory for plausible reasoning. The former, given a set of possible events, explicitly tries to model the probability of occurrence of such events. A comprehensive and elegant mathematical theory strongly supports Bayesian theory for probability and is widely used in science. When no exact model is available, the probability of each event is estimated, often by measuring the relative occurrence of each of those.

The Dempster-Shafer theory, instead, aims at modeling the plausibility in reporting the occurrence of a particular event. In simple words, by using the Dempster-Shafer theory, it is possible to represent how likely an event has happened, according to the fact that it has been reported by someone. What is measured, then, is the reliability assigned to whom reports the occurrence of an event when reporting it, instead of the probability of the event itself. According this formulation, considering the case of classifiers used for attack detection, we don't evaluate the probability of occurrence of either normal or anomalous behavior by the users. Instead, we model the reliability of a classifier when it reports a specific event. In an environment as variable and changing as a network, in fact, it would be really difficult to exactly and permanently model the probability of occurrence of hardly any event. By using the Dempster-Shafer, then, we only have to deal with the problem of modeling the expected behavior of the used classifiers, and the reliability associated to each of their outcomes. As it's easily understood, the latter problem is far less complicated to deal with, and more realistic to solve, than the problem of exactly modelling the behavior of every possible type of user of a network.

3.4.1 The Frame of Discernment

Instead of dealing with a set of events, the Dempster-Shafer theory is founded on the concept of *frame of discernment*. Usually, the frame of discernment θ consists of M mutually exclusive and exhaustive hypotheses A_i, $i = 1, \ldots, M$. Any subset $\{A_i, \ldots, A_j\} \subseteq \theta$ represents an hypothesis. As the number of possible subsets of θ is 2^θ, the generic hypothesis is an element of 2^θ. In our case, we only consider two hypotheses (classes), namely *Normal* and *Attack*; hence, the frame of discernment is

$$\theta = \{\{Normal\}, \{Attack\}\}$$

and

$$2^\theta = \{\{Normal\}, \{Attack\}, \{Normal, Attack\}\}$$

whereas in the Bayesian case only the events $\{\{Normal\}, \{Attack\}\}$ would be considered.
$\{Normal\}$ and $\{Attack\}$ are referred to as *simple events* or *singletons*, while $\{Normal, Attack\}$ as *composite event*.

3.4.2 bpa for Labeling Reliability

According to the D-S theory, a *basic probability assignment* (*bpa*) can be associated to each base classifier, which describes the subjective degree of confidence attributed to it. What is modeled, then, is not the analyzed phenomenon, but the belief in what the base classifiers report about it.
When assigning a *bpa*, there are some requirements which have to be met. They descend from the fact that the *bpa* is still a probability function, hence has to respect the constraints for mass probability functions. Each *bpa* is such that $m : 2^\theta \to [0, 1]$, where θ indicates the so called *frame of discernment*. Furthermore, also the following properties have to hold:

$$m(\varnothing) = 0 \qquad\qquad \sum_{A \subseteq 2^\theta} m(A) = 1$$

The aim of assigning a *bpa* is to describe the reliability of a particular classifier in reporting a specific event. Such a representation is suitable for combination, but as we want to deal with combined results in the same way, we also impose the constraint that the combination of several *bpa* by means of the D-S rule still has to be a *bpa*. The uncertainty in the final decision will be inversely proportional to the extent to which the base classifiers agree.

3.4.3 From bpa to Class Labels

In the next subsection we will present some general criteria for assigning *bpa*'s to different categories of classifiers. Now, we want to illustrate first how it is possible to define a criterion for obtaining a *crisp* classification of each packet from the overall *bpa* calculated after combination, by means of a properly defined index. *bpa*'s in fact, only give indication regarding the overall degree of confidence in reporting the occurrence of each of the known events, while we are interested in knowing whether one of such events, has been detected or not. The function which allows us to transform the *bpa* in a detection result descends from observation of the relations between *Belief*, *Plausibility* and *Uncertainty*. Let $A, B \in \theta$; according to definitions:

$$Bel(B) = \sum_{A \subset B} m(A) \tag{3.1}$$

$$Pls(B) = \sum_{A \cap B \neq \emptyset} m(A) \tag{3.2}$$

$$Unc(B) = Pls(B) - Bel(B) \tag{3.3}$$

In the two-event case, we observe that

$$\begin{aligned} Bel(\{Normal\}) &= m(\{Normal\}) \\ Pls(\{Normal\}) &= m(\{Normal\}) + m(\{Normal, Attack\}) \end{aligned}$$

$$\begin{aligned} Bel(\{Attack\}) &= m(\{Attack\}) \\ Pls(\{Attack\}) &= m(\{Attack\}) + m(\{Normal, Attack\}) \end{aligned}$$

Hence,

$$\begin{aligned} Unc(\{Normal\}) = Unc(\{Attack\}) &= \\ &= m(\{Normal, Attack\}) \end{aligned}$$

As stated before, we aim at defining an index which allows us to associate a single *crisp* label to each packet, even though the chosen packet labeling mechanism inherently provides *soft* [55] labels.
So, what we have to define is a *reliability index RI* which tells us how much difference is there between the degree of belief the system has in stating each of the possible hypotheses. Furthermore, such a reliability index should preferably have a direct dependency on the difference between the degrees of belief in each of the two simple hypotheses, and an inverse dependency on

the degree of belief in the composite hypothesis, which explicitly measures the uncertainty associated to each decision. So, we defined RI as:

$$\begin{aligned}
RI &= \frac{Bel(\{Attack\}) - Bel(\{Normal\})}{1 + Unc(\{Attack\})} = \qquad (3.4)\\
&= \frac{Bel(\{Attack\})}{1 + Unc(\{Attack\})} - \frac{Bel(\{Normal\})}{1 + Unc(\{Normal\})} = \\
&= \frac{m(\{Attack\})}{1 + m(\{Normal, Attack\})} - \frac{m(\{Normal\})}{1 + m(\{Normal, Attack\})} \qquad (3.5)\\
&= \frac{m(\{Attack\}) - m(\{Normal\})}{1 + m(\{Normal, Attack\})}
\end{aligned}$$

The index RI is defined in such a way that, if its value is $+1$, there is the highest reliability in detecting an ongoing malicious activity; if it is -1, it is quite sure that the observed traffic is normal; if it is 0, there's the maximum uncertainty, hence the packet will should be rejected.
In the first case, in fact,

$$m(\{Attack\}) = 0$$

while

$$m(\{Normal\}) = 1$$

and

$$m(\{Normal, Attack\}) = 0$$

In the second case, the opposite scenario is verified, as

$$m(\{Attack\}) = 0$$

while

$$m(\{Normal\}) = 1$$

and

$$m(\{Normal, Attack\}) = 0$$

In the latter case instead,

$$m(\{Normal\}) = 0$$

$$m(\{Attack\}) = 0$$

and

$$m(\{Normal, Attack\}) = 1$$

In order to reject unreliably labeled packets, the value of RI can be compared with a threshold τ (see figure 3.1), which is related to the desired classification reliability. If it falls outside the range $[-\tau,\tau]$ described by such threshold, then the packet can be classified, with at least the specified degree of confidence; otherwise, the packet will fall inside an *uncertain* region (*guard region*). In that case, it will be rejected and eliminated from the labeled database we are interested to build. Note that the higher the threshold, the more reliable the classification of each packet, the higher the number of rejected packets.

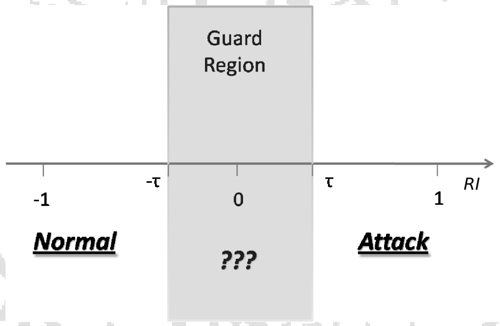


Figure 3.1: From RI to packet labeling or rejection as a function of τ

3.4.4 The Dempster-Shafer Combination Rule

Based on the bpa function $m(A)$, it's possible to calculate the overall belief in the events represented by the set A as:

$$\text{bel}(A) = \sum_{B \subseteq A} m(B) \tag{3.6}$$

where B represent every possible subset of A. Obviously, for each $A_i \subseteq \theta$, the properties $bel(A_i) = m(A_i)$ and $bel(\theta) = 1$ hold. Besides a rigorous formulation to represent evidences, Dempster and Shafer propose a method for combining several sources of information referring to the same frame of discernment θ. Let's assume, for simplicity's sake, we are willing to combine two different sources of information. As stated before, each source of information is represented in terms of the belief associated to it, when reporting specific events. In our case, the degree of belief is expressed by the basic probability functions m_1 and m_2, defined over the same domain θ. The operation of combining evidences from multiple sources is usually represented as an orthogonal sum, that is to say $m = m_1 \oplus m_2$. More in general, the method for combining evidences coming from multiple sources is named *Dempster-Shafer combination rule*, and is defined as:

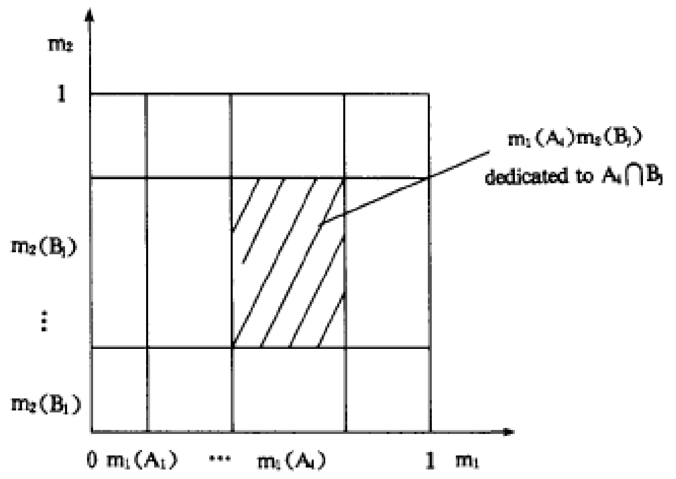


Figure 3.2: Orthogonal Sum

$$\mathrm{m}(A) = k^{-1} \sum_{A_1 \cap A_2 = A} m_1(A_1) m_2(A_2) \tag{3.7}$$

where

$$k = \sum_{A_1 \cap A_2 \neq \emptyset} m_1(A_1) m_2(A_2) \tag{3.8}$$

Starting from 3.7 on the preceding page and 3.8 on the previous page, and due to the associative property of the sum operation, the combination rule can be easily extended to an arbitrary number of information sources, resulting in an arbitrary number of different bpa's. As an example, let's consider the information sources whose degree of belief is expressed by the bpa's $m_1, m_2, \ldots, m_n$. The bpa resulting from their combination will be formally defined as:

$$\mathrm{m} = \bigoplus_{i=1}^{n} m_i \tag{3.9}$$

which translates, in formulas, in:

$$\mathrm{m}(A) = k^{-1} \sum_{\bigcap_i A_i = A} \prod_{1 \le i \le n} m_i(A_i) \tag{3.10}$$

The normalization factor k in equation 3.10 is:

$$k = \sum_{\bigcap_i A_i \neq \emptyset} \prod_{1 \le i \le n} m_i(A_i) \tag{3.11}$$

It is worth observing that the normalizing factor k is independent from any specific value of A. The value k can therefore be considered a constant, once the bpa's are fixed.

3.5 An Architecture for Automatic Packet Labelling

In network security, there are not too many issues in collecting large amounts of traffic which might be used to train a supervised classifier. Unfortunately, such traffic often lacks a reliable labeling. This gives raise to a paradox: if we want to properly train a classifier, we need to have correctly labeled network traffic; but, if we have such labeled traffic, maybe we already know how to exactly discriminate between normal and malicious behavior. Hence, we already know how to tell attack packets apart from normal ones, thus we don't need a classifier.

In order to overcome such a paradox, we propose an architecture for automatically building up a traffic database, containing packets each labeled either as *normal* or as an *attack*. The proposed architecture (see figure 3.3) consists of several base classifiers, called Base IDS (B-IDS in the following). According to the definition of our framework, B-IDS must not require to be trained on labeled data. Typical examples are signature-based IDS (such as Snort$^{\text{TM}3}$) or, more in general, any IDS based on unsupervised techniques. The bank of B-IDS starts analyzing offline the packets contained in a dumped traffic database. As we propose to use multiple detection techniques, it is necessary

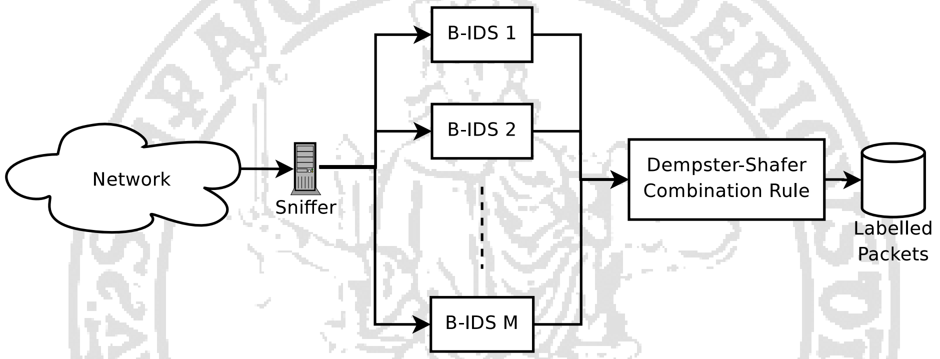


Figure 3.3: B-IDS and the D-S combination rule

to employ some fusion strategy in order to combine their outputs. Since we have no prior knowledge of the traffic characteristics, we use the Dempster and Shafer [44] combination rule, which can completely disregard any prior knowledge about the data to be classified. Unfortunately, some of the types of B-IDS chosen for the first stage have no capacity to automatically evolve. The signature based ones, have to receive completely new signatures in order to learn how to recognize new types of attack. The unsupervised ones, instead, need new training. Since the traffic database in use is the one whose packets we want to associate a label to, new training on the same packets will probably lead to poor or eventually no improvement at all. Hence, we add a second stage which, due to its training strategy, can bring new knowledge to the whole system, and hopefully improve its labelling capabilities. The

[3]http://www.snort.org

components involved in the *evolved* system at the second stage are named *Supervised Intrusion Detection Systems*, or *Slaves*; in short, in the following we'll refer to them as *S-IDS*. The sistem, after the introduction of the new S-IDS modules, will look like the one depicted in figure 3.4. For the sake of

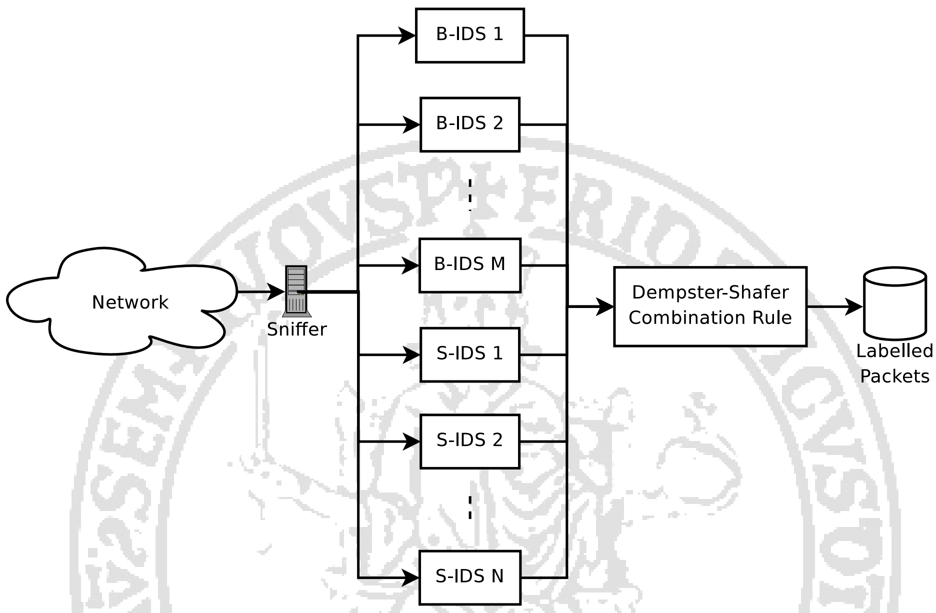


Figure 3.4: D-S based architecture for packet labelling

synthesis, in figure 3.5 on the next page we sketch the whole operating cycle of the packet labelling system. First, a preliminary labelling is obtained, for variuos values of τ. Then, all the S-IDS are trained and, when training is over, integrated with the rest of the system. After integration, the previous step is iterated, until the packet labeling process converges. In the following we will illustrate the details involved in all the described phases, and introduce a terminating criterion for the iterative incremental labelling process.

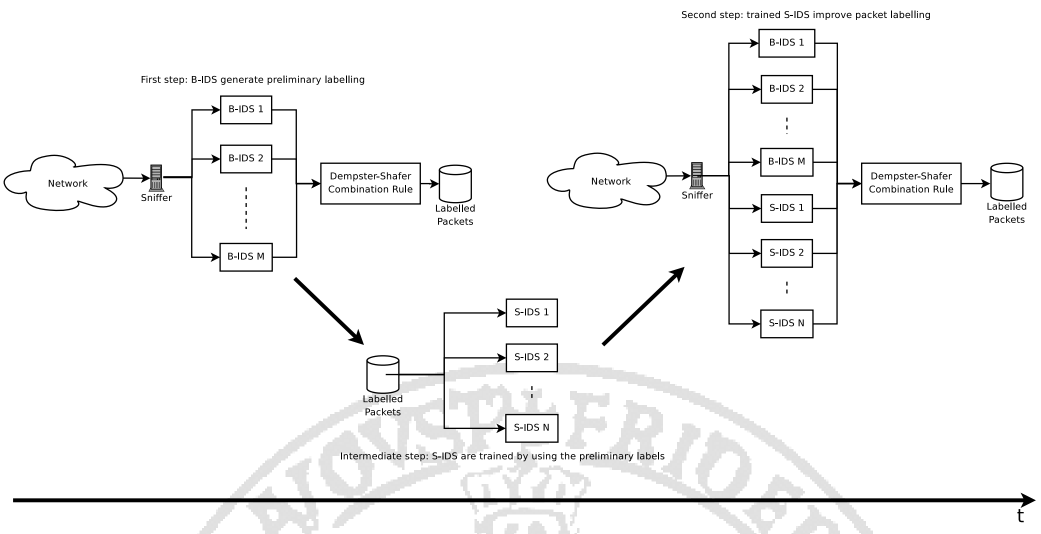

Second step: trained S-IDS improve packet labelling
First step: B-IDS generate preliminary labelling
Network
Sniffer
B-IDS 1
B-IDS 2
B-IDS M
Dempster-Shafer Combination Rule
Labelled Packets
Network
Sniffer
B-IDS 1
B-IDS 2
B-IDS M
S-IDS 1
S-IDS 2
S-IDS N
Dempster-Shafer Combination Rule
Labelled Packets
Labelled Packets
S-IDS 1
S-IDS 2
S-IDS N
Intermediate step: S-IDS are trained by using the preliminary labels
t

Figure 3.5: Evolution in time of the D-S based system

3.5.1 The First Stage – Base Intrusion Detection Systems

Assigning bpa's to B-IDS

Usually, the D-S combination rule is referred to as a technique which requires training at fusion level [57]. In fact, the *bpa*'s are usually assigned according to the so-called *confusion matrix* introduced in section 2.1.3 on page 28, which describes the performance of a specific classifier on a given training set [115]. Hence, the confusion matrix is stricly related to both the classifier's characteristics, and the dataset it's been trained on. In our case, we don't have any training data available, hence we use a different method for assigning a *bpa* to each base classifier. In particular, we assign them *bpa*'s which are related to the typical performance of the detection technique they implement. Then, starting from the category a base classifier belongs to (i.e., anomaly vs. misuse detection, unsupervised vs. supervised, etc.), we define some criteria for assigning a suitable *bpa*. In particular, we assign a higher value to $m(\{Attack\})$ if a signature-based misuse detector raises an alert. Such systems, in fact, are built to correctly detect the malicious behaviors described by the known signatures. On the other hand, the absence of alerts by a signature-based misuse detector is not necessarily related to the absence of any anomalies. A modified attack, in fact, can easily evade such detection technique. Thus, we assign low values to $m(\{Normal\})$ when such systems notify this kind of traffic.

More formally, if we denote with $m^{sb}_{A|A}$ the value $m(\{Attack\})$ attributed to the evidence of an attack when a signature-based IDS attributes a packet to the *Attack* class, and with $m^{sb}_{N|N}$ the value $m(\{Normal\})$ attributed to the evidence of normal behavior when the same IDS does not declare the presence of an attack, we have:

$$m^{sb}_{A|A} > m^{sb}_{N|N} \tag{3.12}$$

Anomaly based signature detection systems, instead, have a dual behavior with respect to misuse detectors. On the other hand, detectors based on Pattern Recognition techniques generally attain worse performance in correctly detecting malicious behaviors, but also prove their ability to detect modified or novel attacks. For that reason, we assign lower belief in their classification with respect to signature based detectors, when an attack is detected. Moreover, an higher belief is assigned when a packet is attributed to the normal class with respect to the case in which it is attributed to the attack class.

By using the same notation of equation (3.12), with *ad* denoting an anomaly-based IDS, we can write in this case:

$$m^{ad}_{N|N} > m^{ad}_{A|A} \tag{3.13}$$

$$m^{ad}_{A|A} < m^{sb}_{A|A} \tag{3.14}$$

Thus, each decision from a B-IDS will be supported by the *bpa* associated to it, which will express the degree of belief in the classifier decision. Such *bpa*'s will be combined by using the Dempster-Shafer combination rule [44], in order to obtain a *bpa* for each possible class the packet can belong to. In section 3.6 on page 88 we will show how to assign reasonable value to the *bpa* of each chosen B-IDS, according to the above described criteria.

3.5.2 Preliminary Labelling

Once a *bpa* is assigned to each B-IDS, it is possible to start the first operating phase of our system. As stated before, each packet in the raw traffic trace will be analyzed by each of the B-IDS. In some cases, B-IDS are implemented by means of common, popular intrusion detection software, capable of analyzing raw traffic traces, and reporting the detection of specific events on the network. In some other cases, B-IDS are implemented by means of unsupervised pattern recognition techniques. In the latter case, a preprocessing phase is required, which transform raw packets in a format which is manipulated by such techniques. In our case, it is necessary to extract representative features from network traffic, in order to process them by means of the selected algorithms. Such features are usually calculated either at packet or flow granularity. Usually, features are updated at each packet's arrival, in order to describe the evolution of properties specific of the traffic flow it belongs to. In our scheme, we will consider the preprocessing phase embedded in B-IDS operation, hence it's not explicitly represented either in figure 3.3 on page 74 or 3.4 on page 75. Once a threshold τ is fixed for the specific analysis of the raw packets, the analysis is performed. All the results coming from B-IDS are then collected and combined by means of the Dempster-Shafer block. After that, the packed such that $|RI| > \tau$ are stored, and used for further processing. In this case, the further processing consists in the training of the classifiers employed for the second stage of the system. Obviously, such system will have different performance, depending on the choice of the value of τ.

3.5.3 The Second Stage – Supervised Intrusion Detection Systems

Integrating a Supervised-IDS (S-IDS) in the Architecture

As stated earlier in this chapter, the labeling performed by means of the system described above can be used for training supervised pattern recognition-based IDS. Such IDS should perform better in real environment with respect to unsupervised IDS. As a matter of fact, they have a better generalization power, thus allowing the detection of novel anomalous events, though keeping good performance in detecting well known attacks. Anyway, once we have obtained a *trained supervised* IDS (S-IDS), the question arises whether it is possible to integrate it in the proposed architecture, by combining its outputs with those provided by the bank of B-IDS, and which could be the possible improvements in the detection of attacks. So, this section is devoted to show how an S-IDS can be integrated in the described architecture. In order to be actually able to do that, we need to find an answer to several questions.

First of all, let us recall that an S-IDS obviously needs a training phase. In our case, for the training we'll use the results of the classification performed by the B-IDS. In the previous section we introduced the threshold τ on the value of RI, which determines the degree of reliability of packet labeling. Different threshold values give rise to sub-datasets consisting of different fractions of the whole dataset, which can be considered reliable enough for training according to the criterion driving the choice of the value of τ. To this extent, we have to understand how to select the optimal value for τ, which allows us to obtain the best trade-off between reliability of the labeling and representativeness of the portion of the dataset selected for training. In other words, we would like the best reliability level and the biggest possible fraction of the whole dataset, in order to have a well representative database for training an S-IDS.

Secondly, we must be able to understand how good the training of the S-IDS is at each stage, i.e how to measure the performance of an S-IDS, since we do not know the real labels, but only the ones provided by the ensemble of B-IDS. Furthermore, since we want to integrate an S-IDS in the whole architecture, we need to find a way to define a degree of confidence in the outcomes of such S-IDS, and then the corresponding *bpa* for it.

After answering all these questions, the integration becomes possible; we will show that the obtained classification results can improve if we exploit the

contributions coming from the trained S-IDS.
By using the new assigned labels, then, it is possible to re-train the S-IDS and to combine it again with the ensemble of B-IDS, in order to further improve the labeling. The problem here is to understand when to stop such an iterative process. Furthermore, we want to understand whether it is both useful to re-estimate the *bpa* of the B-IDS's once some S-IDS's are trained and can help in obtaining a more precise labeling of the packets.
The main steps of the procedure for adding an S-IDS to the proposed architecture, aimed at obtaining a more reliable labeling, can be summarized as follows:

- Train the S-IDS using a the labeled dataset obtain for a certain value for the reliability threshold τ
- Evaluate the performance of the trained S-IDS (see Section 3.5.4 on the following page)
- Calculate the *bpa* for the trained S-IDS (see Section 3.5.5 on page 84)
- Combine both B-IDS and S-IDS and evaluate their performance (definition of Estimated Error Rate) (see Section 3.5.6 on page 86)

In the following we sketch the algorithm that can be used for integrating an S-IDS and for evaluating the performance of the whole architecture, having chosen a specific value for the threshold τ.

```
Insertion_S-IDS($B\text{-}IDS\_List, S\text{-}IDS, Traffic, level, \tau$)

 1  if $level = 0$
    ▷ $level$ is 0 if this is the first time the $S\text{-}IDS$ is trained
 2     then $Labels \leftarrow$ D-S_FUSION($B\text{-}IDS\_List, Traffic$)
 3     else $Labels \leftarrow$ D-S_FUSION($B\text{-}IDS\_List, S\text{-}IDS, Traffic$)

 4  $ReliableLabels \leftarrow$ EXTRACT_L($Labels, \tau$)
 5  $ReliableTraffic \leftarrow$ EXTRACT_T($Traffic, \tau$)
 6  $S\text{-}IDS \leftarrow$ TRAIN($S\text{-}IDS, ReliableLabels, ReliableTraffic$)
 7  $NewLabels \leftarrow$ CLASSIFY($S\text{-}IDS, Traffic$)
 8  $bpa[S\text{-}IDS] \leftarrow$ EVAL_S-IDS_PERF($NewLabels, Labels$)
 9  $NewLabels \leftarrow$ D-S_FUSION($B\text{-}IDS\_List, S\text{-}IDS, Traffic$)
10  $EER \leftarrow$ EVAL_PERF($NewLabels, Labels$)
    ▷ EER is the Estimated Error Rate provided
    ▷ by the combination of B-IDS's and S-IDS

11  return $EER$
```

This algorithm should be executed for each possible threshold value, in order to select the one which allows to obtain the best performance. Moreover, the insertion process of an S-IDS in the architecture can be re-iterated by using the newly computed labels. In this case, nothing ensures that the threshold value selected at the previous iteration (*level*) will lead to the best possible system performance. In practice, we have to perform a Breadth-First Search in a tree, by searching the node which gives rise to the best system configuration in terms of EER.
An exhaustive search of the optimal solution would be NP-hard, hence we decided to pursue a suboptimal, though computationally affordable, search strategy, as described in Section 3.5.6 on page 86. At each level, we only consider the node with the best EER and those nodes exhibiting an EER whose value is within an order of magnitude with respect to the lowest. In Section 3.6 on page 88 we will show the results obtained by using this strategy. In the following paragraphs we will detail how it is possible to perform the steps described in the above reported algorithm. Finally, note that since the proposed architecture will provide a soft label for each packet, it should be useful to define a measure for giving a quantitative evaluation of the quality of a performed labeling. Such a definition will be given in Section 3.5.7 on page 87.

3.5.4 Evaluating S-IDS Performance

The problem of assigning a *bpa* to each S-IDS is strictly related to defining the extent to which we rely on the outcomes of such classifiers, whenever they are reporting a specific event. In our case, due to the presence of the reliability index RI, we don't have crisp outputs by the classifiers. Usually, in presence of binary classifiers the *confusion matrix* [37], introduced in 2.1.3 on page 28, is a good means of evaluating how good a classifier performs on a given dataset. In our case the output of each classifier is a *soft* label, hence we have to redefine the confusion matrix for classifiers with soft outputs. Moreover, we do not know the true class of a packet, but we have only the guessed class provided by the combination of the outputs of a set of IDSs. Each instance I of the data to be classified can be associated to an element of the set $\{Positive, Negative\}$, or $\{P, N\}$ for the sake of brevity. The classifier associates to each instance I its prediction regarding the class such an instance belongs to. In order to distinguish between the actual class labels and those assigned by the classifier, we'll use a twofold notation, adding a hat above the latter. Hence, the actual labels will be indicated as P and N, while those assigned by the classifier will be indicated as $\hat{P}$ and $\hat{N}$. Given this notation, we have four possible combination of actual and assigned labels, which are sketched in table 3.5. In our case we have a set of packets,

	Assigned Class	
True Class	$\hat{P}$	$\hat{N}$
P	TP	FN
N	FP	TN

Table 3.5: Confusion matrix for one class classification

labeled using the B-IDS part of our proposed architecture; we know that each packet has an associated reliability index, which we denote with RI. The closer to 1 RI is, the more confident we are in assigning the packet to the *Attack* class. On the other hand, the closer the index is to -1, the more we are confident in saying that the packet can be assigned to the $Normal$ class. Furthermore, for values of RI close to 0, there is a high uncertainty in assigning the packet to either of the two classes. First of all, for each packet, let us assume that it belongs to the P class if $RI > 0$ and to the N class otherwise. The same packet will be attributed to $\hat{P}$ if the considered ensemble of IDS recognizes it as an *Attack*, or to $\hat{N}$ otherwise.

Then, let us recall that our aim is to evaluate the S-IDS performance, in order to assign a *bpa* to it. We want to have classifiers able to correctly classify all the packets characterized by a value of RI close to either -1 or 1. The focus is not really on packets whose value of RI is close to 0, since those are inherently characterized by a high degree of uncertainty. If the system is not able to correctly classify such packets, it is acceptable that they are ex-

plicitly *rejected*. What we want to do is to weight differently the outcome of each classifiers, according to the value of RI. That's why we introduce the concept of a *Weighted Confusion Matrix*.
Let $\mathbf{RI} = (RI_1, RI_2, \dots, RI_m)$ be the vector of all the different reliability values associated to the packets. Starting from the confusion matrix reported in table 3.5 on the preceding page, which is drawn again for the sake of clarity, we can define the *true positive rate* (tp), the *false positive rate* (fp), the *true negative rate* (tn) and the *false negative rate* (fn) as follows:

$$tp = \frac{TP}{P} \qquad fp = \frac{FP}{N}$$

(3.15)

$$tn = \frac{TN}{N} \qquad fn = \frac{FN}{P}$$

Now we will show how it is possible, for example, to derive a *weighted true positive rate*, say $tp_{\mathbf{RI}}$, which is function of the vector $\mathbf{RI}$. The same considerations can be easily extended to the other quantities reported in equations 3.15. Let's first consider how we can rewrite tp:

$$\begin{aligned} tp = \frac{TP}{P} &= \frac{TP_{RI_1} + TP_{RI_2} + \dots + TP_{RI_m}}{P} = \\ &= \frac{\frac{TP_{RI_1}}{P_1}P_1 + \frac{TP_{RI_2}}{P_2}P_2 + \dots + \frac{TP_{RI_m}}{P_m}P_m}{P} = \\ &= \frac{(tp_{RI_1})P_1 + (tp_{RI_2})P_2 + \dots + (tp_{RI_m})P_m}{P} \end{aligned}$$

where TP_{RI_i} represents the true positive having a reliability index equal to RI_i and P_i is the number of the packets whose reliability is RI_i.
Now, by substituting tp_{RI_i} with $tp_i = w(RI_i) \cdot tp_{RI_i}$ we obtain a true positive rate weighted by a function of the reliability index, namely $w(RI_i)$. So, we finally arrive at the definition of $tp_{\mathbf{RI}}$:

$$\begin{aligned} tp_{\mathbf{RI}} &= \frac{(w(RI_1))(tp_{RI_1})P_1 + \dots + (w(RI_m))(tp_{RI_m})P_m}{P} \\ &= \frac{TP_{\mathbf{RI}}}{P} \end{aligned}$$

The choice of the weight function can either slow down or speed up the convergence of the training algorithm for an S-IDS. Next step in the process of defining the D-S based architecture is the definition of the weighting function we've been discussing so far. As stated before In order to choose a suitable function, let us first consider that the weight is supposed to increase together with the value of the reliability index. Hence, by recalling that $0 \leq RI_i \leq 1$, it must be:

$$RI_i = 0 \Rightarrow w(RI_i) = 0$$
$$RI_i = 1 \Rightarrow w(RI_i) = 1$$

Since for a reliably labeled packet i it is $RI_i = 1$, such a piece of information is worth the maximum relative weight possible. On the other hand, for packets labeled with very low reliability we know that $RI_i \approx 0$, hence such a contribution is supposed not to influence the final decision.
It is reasonable, given these considerations, to choose the identity function for our purposes:

$$w(RI_i) = RI_i$$

. Obviously, it might be possible to choose several different functions, thus changing the distribution of weights according to the value of the reliability index and affecting the convergence time for S-IDS training. This will be matter of future investigations.

3.5.5 From Weighted Confusion Matrix to bpa

The method we adopt to evaluate the *bpa* starting from the modified version of the confusion matrix relies on three parameters: the *Recognition Rate* (r), the *Substitution Rate* (s) and the *Rejection Rate* with respect to the possible classes (in our application, just *Normal* and *Attack*). Such parameters can be obtained from the confusion matrix of a classifier. In particular, as shown in [115], given a generic class C, the corresponding *bpa*, and the confusion matrix, will be:

$$\begin{aligned} m(C) &= r \\ m(\neg C) &= s \\ m(\Theta) &= 1 - r - s \end{aligned}$$

With respect to a given class C, r is the percentage of samples assigned to class C by the classifier, and actually belonging to such a class; s is the percentage of samples belonging to class C but assigned by the classifier to some other class; by rejection rate, finally, we mean the percentage of samples which were rejected, and hence not assigned to any class [36]. The S-IDS we are using at the moment don't implement rejection explicitly. Yet, by observing the values of the weighted confusion matrix, we can see that, due to the weighting operation:

$$\begin{aligned} tp_{\mathbf{RI}} + fn_{\mathbf{RI}} &\leq 1 \\ tn_{\mathbf{RI}} + fp_{\mathbf{RI}} &\leq 1 \end{aligned}$$

That stated, the *bpa* assigned to the S-IDS after its training will be:

- when the S-IDS attributes a packet to the *Attack* class:

$$\begin{aligned} m(\{\mathit{Attack}\}) &= tp_{\mathbf{RI}} \\ m(\{\mathit{Normal}\}) &= fn_{\mathbf{RI}} \\ m((\{\mathit{Normal,Attack}\})) &= 1 - tp_{\mathbf{RI}} - fn_{\mathbf{RI}} \end{aligned}$$

- when the S-IDS attributes a packet to the *Normal* class:

$$\begin{aligned} m(\{\mathit{Attack}\}) &= fp_{\mathbf{RI}} \\ m(\{\mathit{Normal}\}) &= tn_{\mathbf{RI}} \\ m((\{\mathit{Normal,Attack}\})) &= 1 - tn_{\mathbf{RI}} - fp_{\mathbf{RI}} \end{aligned}$$

The values of the weighted confusion matrix are those obtained after training the S-IDS with the labeled packets computed by the ensemble of B-IDS.

3.5.6 Terminating the S-IDS Training Process

After the first training iteration, we obtain new labels for the packets. By combining the outcome of each S-IDS with the outcome the B-IDS ensemble, we obtain a modified and hopefully more accurate labeling for the dataset. After the first step, we evaluate the *bpa* for each of the S-IDS again, in an iterative fashion.

In order to define a termination criterion, we introduce the concept of *Estimated Error Rate* (EER):

$$EER = \frac{FP_{\mathbf{RI}} + FN_{\mathbf{RI}}}{N + P}$$

where $N + P$ is the total number of packets and $FP_{\mathbf{RI}} = fp_{\mathbf{RI}} \cdot N$, while $FN_{\mathbf{RI}} = fn_{\mathbf{RI}} \cdot P$. In order to iterate the training phase, we can decide to use, for each S-IDS, the configuration associated to the lowest EER (say EER^{best}). Yet, this approach might lead to suboptimal results during iterations since we are using estimated quantities. For this reason we also consider some S-IDS training configurations which lead to higher EER, since during the following iterations they might lead to better overall results. In particular, we keep all the training configurations associated to a value for EER such that:

$$\frac{EER^{best}}{EER} \geq 0.95$$

At each iteration we use an exhaustive procedure, by training each S-IDS with all the possible datasets, i.e. all the dataset obtained by rejecting all the packets characterized by reliability values below a given threshold τ.

Since the process is iterative, we need a termination criterion in order to understand when to stop it. To such purpose, we measure the value of EER^{best} at each iteration. In general, it should be decreasing along the iterations, finally leading to an asymptotically low value. The condition we evaluate, then, is:

$$EER_k^{best} \leq \delta$$

where:

EER_k^{best} is the estimated error rate for each S-IDS, with respect to the $k-th$ iteration. More precisely, it is the minimum error rate among all the chosen configurations for each training iteration.

δ is a threshold value, chosen according to the specific application and the required reliability level.

3.5.7 Evaluating the Labeling Process

In this section we will define a method for evaluating how much the labels computed via the proposed system differ from the actual traffic characterization. We want to compare the *real* traffic labels with those obtained through the labeling process described so far. As stated earlier, the proposed system associates a reliability index, RI, ranging from -1 to 1, to the class label attached to each packet. For coherence's sake, we assigned the values 1 and -1 respectively to packets whose real label is either *Attack* or *Normal*; performance is numerically evaluated via the following quality index:

$$QI = \frac{1}{n} \cdot \sum_{i=1}^{n} |RI_i - C_i|$$

where:

n is the total number of packets of the database

RI_i is the value of the reliability index associated to the $i - th$ packet

C_i is the actual class of the $i - th$ packet; i.e. it is equal to

- 1 for packets belonging to the *Attack* class
- -1 for packets belonging to the *Normal* class

As QI approaches 0, the dataset labeled by means of the proposed system tends to the dataset with the actual crisp labels.

3.6 Experimental Results for Automated Traffic Labelling

In order to test the proposed approach, we used one day of the DARPA 99 traces (more than $1e6$ packets; among them about $1,000$ are attacks). Even if the collection of these traffic traces has been criticized, they still represent the most widely used public dataset, and then can be considered a significant benchmark for testing our architecture. To extract representative features from the raw traffic traces, as required by PR-based systems, we used the preprocessor plug-in for Snort^{TM} described in chapter 2, and freely available on SourceForge [4]. It allows us to extract 26 features from network traffic. Such features derive from the *intrinsic* and *traffic* features defined by Lee and Stolfo in [64]. More details about the feature extraction process can be found in [34].

As B-IDS, we chose a signature-based IDS, namely Snort^{TM}, and two PR-based anomaly detectors, respectively a Rival Penalized Competitive Learning (RPCL) network and an one-class Support Vector Machine (SVM) [18]. The RPCL has only two neurons; after the unsupervised learning phase, we associate to the *Normal* class the neuron which has been selected as the most representative for the packet under exam the most times; by doing this, we implicitly assume that *normal* packets are in a much higher number than *attack* packets. It is worth observing that, since Snort^{TM} works at packet level, we need to use it as a classifier for unprocessed raw tcpdump traffic. Moreover, since we used the version 2.2.4, we enabled its multiple detection feature. Snort^{TM}can then raise more than one alert for each packet. In this case, each alert is considered as an evidence supporting the event $\{Attack\}$ for the current packet. The performance of Snort^{TM}and of the neural based classifiers on the chosen dataset are reported in table 3.6 on the following page.

Now, we are required to model the degree of belief associated to each base

[4]http://s-predator.sourceforge.net

	FA Rate	MD Rate	Err Rate
Snort^{TM}	0.129%	21.789%	0.164%
RPCL	1.477%	21.997%	1.510%
SVM	0.924%	74.571%	1.044%

Table 3.6: Performance obtained by the chosen B-IDS

classifier reporting each of the observed events. As stated before, we do it by using the basic probability assignment. First of all, let's start by assigning a *bpa* to Snort^{TM}. As shown in table 3.7, we assume that Snort^{TM}is very reliable when reporting malicious activities. The probability that an attack actually occurs when Snort^{TM}reports it, is very high, due to its signature-based nature. According to that, we assigned the value 0.9 to $m(\{Attack\})$ when Snort^{TM} raises an alert. Conversely, aware of Snort^{TM}'s low probabil-

	m({N})	**m({A})**	**m({N,A})**
A	0	0.9	0.1
N	0.6	0	0.4

Table 3.7: *bpa*'s assigned to Snort^{TM}

ity of detecting either a novel or modified attack, we don't really trust it too much when reporting normal activity.
Similarly to the case of Snort^{TM}, we considered the typical behavior of the neural-based classifiers we chose, in order to decide when and how much to trust each of them. Such classifiers, due to their unsupervised nature, are not supposed to perform extremely well on any class of traffic, since they're not meant to be too specialized. Yet, on the other hand, they may also be able to detect novel attacks, thus compensating the lack of such ability for Snort^{TM}and similar signature based B-IDS. In table 3.8 on the following page we observe the *bpa*'s assigned to it.

By using such *bpa*'s, and combining the results for the three considered B-IDS, we are able to obtain a preliminary labeling of the raw traffic. To this regard, we verified that there are no differences in such a labeling if the

	m({N})	**m({A})**	**m({N,A})**
A	0	0.5	0.5
N	0.6	0	0.4

Table 3.8: *bpa*'s assigned to RPCL and SVM

τ	**FA Rate**	**MD Rate**	**Err Rate**	**Rej Rate**
0.0	0.278%	21.789%	0.313%	0%
0.3	0.027%	21.945%	0.063%	0.251%
0.6	0.015%	74.623%	0.051%	2.404%
0.9	0.015%	74.623%	0.015%	99.943%

Table 3.9: Performance obtained by combining the chosen B-IDS as a function of the threshold τ

above reported *bpa*'s vary within a 5% range and the constraints of equations (3.12)-(3.13) still hold.
By varying the value of the reliability threshold and comparing it to the indicator defined in eq. (3.4), we finally obtain the error rates described in table 3.9. Here, *False Alarm Rate* and *Rejection Rate* are meant in the canonical way. When computing the *Missed Detection Rate*, we consider all the attack packets which were either misclassified as normal, or rejected. In other words, if an attack packet is rejected, we consider it as missed. When evaluating the *Error Rate*, instead, we consider all the packets which are misclassified, regardless the number of rejections.
It can be noted in table 3.9 that if we consider only packets with an *RI* index greater than 0.3, we are able to obtain a better performance with respect the one shown by the best B-IDS. In this case, anyway, about 3,000 packets are not classified.

3.6.1 Training and Integrating an S-IDS

In this section we will describe the *evolution* of an S-IDS. Here we recall that such classifiers always need a training phase performed on pre-classified traffic. In this case, we will first use the traffic labeled by the ensemble of B-IDS

for training, by using certain reliability thresholds.
In the following, we will show an example of S-IDS integration with the rest of the system. As S-IDS, we have chosen SLIPPER [23], a rule-based learning system that creates a rule set by iteratively boosting a greedy rule-builder. To such extent, we'll describe how the whole procedure applies to the case of SLIPPER. We will follow the steps described by the pseudocode in section 3.5.3 on page 78. In figure 3.6 on page 93 we show how different configurations for the threshold are chosen, according to the value of EER, until the termination condition is met. In particular, we considered a δ value equal to $1e - 5$, which is reached at the third training iteration. In the figure are also depicted the *actual* error rate ER and the estimated error rate EER. The former is evaluated on the reference dataset, by using the reference labels. In such a way we are able to evaluate the estimation errors and to justify the optimization strategies we employ.
Note that in the proposed example, during the first iteration we don't choose a single value for the threshold, since there are two possible values associated to low and very similar values for EER. In particular, their ratio is over the 0.95 threshold fixed in Section 3.5.6 on page 86. It is also interesting to note that the results provided by the proposed optimization procedure will actually lead to the best possible performance (i.e. the one obtained if the whole tree would be searched).
In order to have a deeper insight on the above described process, in table 3.10 on the following page we report an example of how the *bpa*'s can be evaluated starting form the *weighted confusion matrix* described in Section 3.5.5. In particular, values in table 3.10 refers to the first training iteration of SLIPPER and to the evaluation of the weighted confusion matrix in case of SLIPPER being trained with the dataset labeled by the ensemble of B-IDS and by considering a threshold $\tau = 0$ (i.e., we consider the whole dataset). By considering the results obtained by combining SLIPPER and the ensemble of B-IDS, we are also able to obtain a much better labeling that the one obtained by considering only the B-IDS's alone. In order to better evaluate

(a) Weighted confusion matrix obtained for SLIPPER

$tn_{\mathbf{RI}}$=0.872	$fp_{\mathbf{RI}}$=0.000
$fn_{\mathbf{RI}}$=0.068	$tp_{\mathbf{RI}}$=0.400

(b) *bpa*'s evaluated starting from the values reported in (a)

	m({N})	**m({A})**	**m({N,A})**
A	0.068	0.400	0.532
N	0.872	0.000	0.128

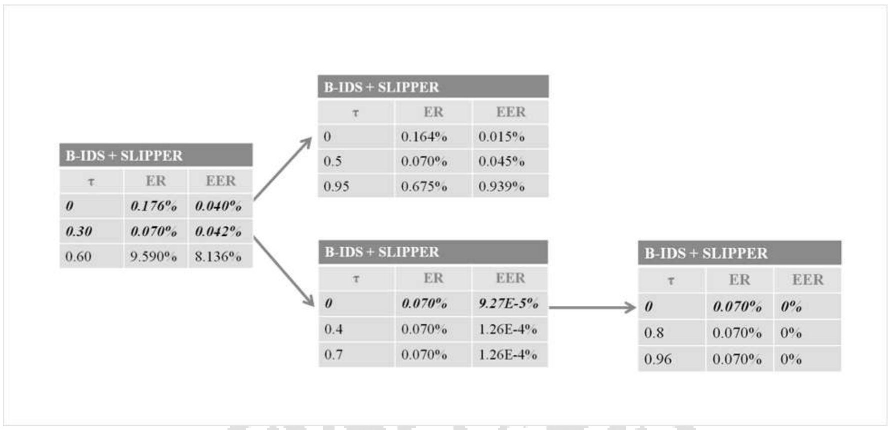
Table 3.10: *bpa*'s assigned to SLIPPER. First training iteration, by using the whole database ($\tau = 0$).

such aspect, let us consider the results reported in table 3.11, where the value 0 for the parameter QI, ideally represents the correctly labeled dataset. In such a table, we show also how the quality of the labeling increases along the various S-IDS training iterations. At this point, we want to show how,

IDS Combination	**QI**
Best B-IDS (Snort^{TM})	0.568
B-IDS only	0.130
B-IDS and SLIPPER (1st training iteration)	0.019
B-IDS and SLIPPER (2nd training iteration)	0.004
B-IDS and SLIPPER (3rd training iteration)	0.001

Table 3.11: Performance obtained by combining the chosen B-IDS with Slipper through the iterative process

through the proposed labeling scheme, it is possible to obtain better training results than the ones obtained by the combination of the B-IDS alone. This would both justify the increased complexity of the architecture resulting in increased requested computational power, coming from the insertion of further classification modules, and the need for multiple iterations over the same data, in order to improve the knowledge of the whole system about the problem at hand. In order to do that, we refer to table 3.12 on the next page, where we show the performance obtained by SLIPPER when it's trained by using the labels provided by the ensemble of B-IDS and the proposed archi-



B-IDS + SLIPPER		
τ	ER	EER
0	***0.176%***	***0.040%***
0.30	***0.070%***	***0.042%***
0.60	9.590%	8.136%

B-IDS + SLIPPER		
τ	ER	EER
0	0.164%	0.015%
0.5	0.070%	0.045%
0.95	0.675%	0.939%

B-IDS + SLIPPER		
τ	ER	EER
0	***0.070%***	***9.27E-5%***
0.4	0.070%	1.26E-4%
0.7	0.070%	1.26E-4%

B-IDS + SLIPPER		
τ	ER	EER
0	***0.070%***	***0%***
0.8	0.070%	0%
0.96	0.070%	0%

Figure 3.6: Results obtained by the proposed architecture at different training iterations. The procedure described in Section 3.5.3 on page 78 is used for searching the configuration leading to the lowest EER

tecture (first two rows). For the sake of comparison, also the results obtained by training SLIPPER with different fractions of the dataset with the real labels are reported (rows 3–6). In this case, the difference with respect to the

		FA Rate	**MD Rate**	**Err Rate**
labels provided by the bank of B-IDS	$\tau = 0$	0.141%	21.997%	0.176%
labels provided by the proposed architecture	$\tau = 0$	0.035%	21.945%	0.070%
actual labels	10%	0.045%	88.820%	0.190%
	20%	0.025%	79.407%	0.154%
	30%	0.004%	19.345%	0.036%
	100%	0.013%	12.117%	0.032%

Table 3.12: SLIPPER performance as a function of the dataset used for training

ideal training set (i.e., the whole database with actual labels) in terms of error rate turned out surely acceptable (only 0.03% with a recognition rate of 99.9%), thus confirming our claims. Moreover, it has to be pointed out that labels provided by our approach allow us to train a classifier that performs better than the one trained with the 20% of the whole database with actual labels, which anyhow exhibits an unacceptable missed detection rate.

We have also conducted a statistical significance analysis with the t-test, as suggested in [55]. Within a significance threshold of 1%, the difference between results obtained by our approach and those obtained by considering the 20% of the whole database with actual labels resulted significant. This is a very good result, since such a dataset is composed by over 230, 000 samples that have been labeled by hand, with a significant waste of time with respect to an automatic labeling.

3.6.2 Key Findings on Data Collection

In this section we deal with the problem of collecting labeled packets, in order to usefully train intrusion detection systems. We describe an architecture based on the Dempster-Shafer theory which, by analyzing raw packets, is able to assign labels to them. It is also able to point out how reliable each of the labels is, thus allowing to choose only the packets which satisfy a certain level of confidence. By training a supervised algorithm with data labeled in this way, we show that it is possible to obtain results which are similar to those obtained by using the actual class of all the packets. Different training set sizes can be indeed obtained by setting a suitable threshold on the index that represents an estimate of the accuracy of the packet labels provided by the proposed method. So, we were also able to observe an interesting tradeoff between the attainable performance and the size of the training set used. The definition of suitable criteria for choosing the best threshold value will be subject of future investigation.
As future developments, we deem it might be possible to extend the D-S based architecture by including also semi-supervised classifiers; moreover, we want to test the whole architecture on other traffic traces. Furthermore, we think it'd be an interesting development to study the impact of the choice of different weighting functions such as the ones described in section 3.5.4 on page 81. Of course, such function might impact both the accuracy of the labeling and the computational resource demand involved in the whole process.

Chapter 4

Multiple Classifier Systems for Network Security

4.1 Combining Genetic-based Misuse and Anomaly Detection

In this section we present a system exploiting genetic algorithms for deploying both a misuse-based and an anomaly-based classifier. Hence, by suitably combining the results obtained by means of such techniques, we aim at attaining a highly reliable classification system, still with a significant degree of new attack prediction ability. In order to improve classification reliability, we introduce the concept of rejection: instead of emitting an unreliable verdict, an ambiguous packet can be logged for further analysis. Tests of the proposed system on a standard database for benchmarking intrusion detection systems are also reported. In general, one of the main drawbacks occurring when using pattern recognition techniques in real environments is the high false alarm rate they often produce [43]. This is a very critical point in a real environment, as pointed out in [6]. In order to realize an IDS that is capable of detecting intrusion by keeping the number of false alarms as low as possible, here we propose a genetic-based system that tries to combine some of the peculiarities of the misuse and of the anomaly detection approaches. In particular, starting from the features proposed in [64], a genetic algorithm is used to generate two distinct sets of rules. The first set is devoted to individuate normal traffic connections (as in an anomaly detection approach), while the second one is suited for detecting specific attacks (following the misuse detection paradigm). A packet is then classified as belonging to an attack if it matches almost one of the rules of the second set and no one of the first set. On the other hand, a packet is labeled as normal if it matches almost one of the rules devoted to detecting normal traffic and no one of those generated for characterizing attacks. In all the other cases, the packet is rejected by the system, since it cannot be correctly classified with an high reliability. This permits to reduce the number of false alarms. Note that reject, in this case, means that the data about a rejected packet are only logged for further processing, without raising an alert for the system manager. It is worth pointing out that other rule-based classifiers have been employed in an intrusion detection system (for example RIPPER [22], used by Lee and Stolfo in [64]). However, they only follow either the misuse or anomaly detection approach, thus giving rise to a false alarm rate that cannot be acceptable in a typical real environment.

4.1.1 A Genetic Approach to Classification Rules Generation

As stated before, the proposed system is rule-based. Two sets of rules are generated, each one devoted to individuate a specific class of traffic, which in our case can be namely either attack or normal. In order to classify a new packet, the results of the two rule-based classifiers are suitably combined by means of a decision engine. In particular, if the feature vector describing a packet and the connection it belongs to matches one of the rules related to the normal traffic and does not match any of the rules related to the attack class, it is attributed to the normal traffic class. Vice versa, if it matches at least one rule describing attacks and no one of the rules describing normal traffic, an alert is raised. In all the other cases, data about the connections are just logged for further processing (see figure 4.1 on the next page). Each set of rules is generated by means of a genetic algorithm based on a particular

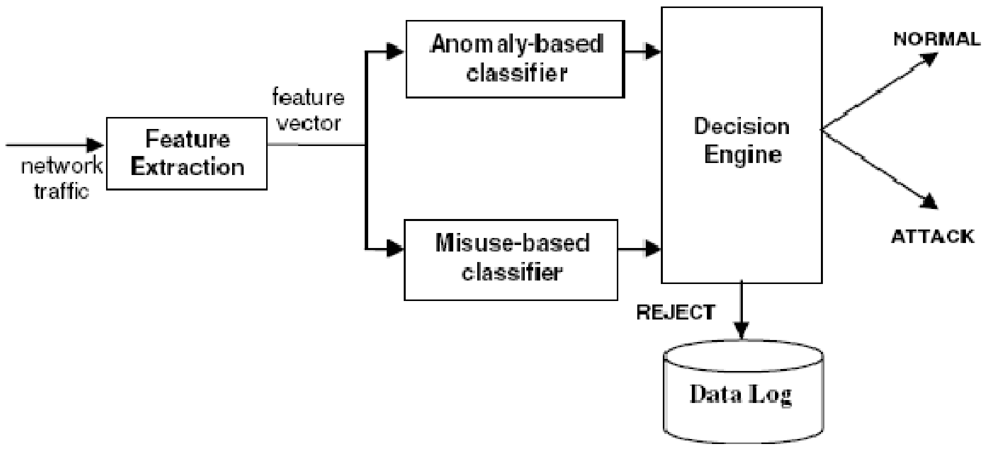


Figure 4.1: Genetic based Misuse and Anomaly detection

structure of the chromosomes. Such a structure was developed for suitably representing the boundaries of the region of the n-dimensional space containing the feature vectors representing the network connection belonging to the class the chromosome refers to. Each chromosome consists of n genes. Each gene is associated to an element of the feature vector to be classified and is composed by a pair of values, $x_{i_{MIN}}$ and $x_{i_{MAX}}$. Such values represent, respectively, the lower and the upper limit of an interval. If the values of all the elements of the feature vector fall within the limits specified by the corresponding genes of a chromosome, this feature vector is attributed to the class the chromosome refers to. The minimum value that $x_{i_{MIN}}$ can assume is $-\infty$, while the maximum value that $x_{i_{MAX}}$ can assume is $+\infty$. The conversion from a rule to a chromosome and vice versa is immediate since they are simply two different ways to represent decision regions (see figure 4.2). Thus, each chromosome represents a hyper-region in the n-dimensional feature space. The aim of the proposed algorithm is to identify the region which the feature vectors belonging to a given class (normal or attack) lie into, that is, to select the corresponding chromosomes. The first step consists in the generation of an initial population of chromosomes, by assigning to each gene pairs of pseudo-random values. The assigned values are randomly selected

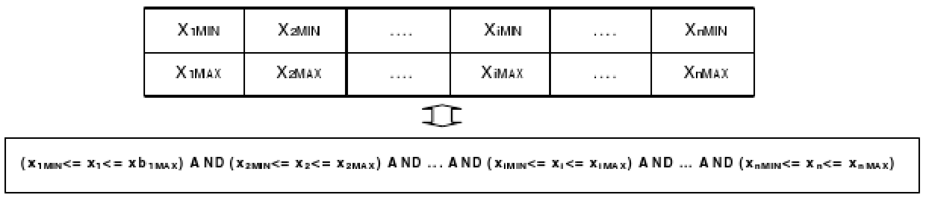

X_{1MIN}
X_{2MIN}
....
X_{iMIN}
....
X_{nMIN}
X_{1MAX}
X_{2MAX}
....
X_{iMAX}
....
X_{nMAX}
$(x_{1MIN} <= x_1 <= xb_{1MAX})$ AND $(x_{2MIN} <= x_2 <= x_{2MAX})$ AND ... AND $(x_{iMIN} <= x_i <= x_{iMAX})$ AND ... AND $(x_{nMIN} <= x_n <= x_{nMAX})$

Figure 4.2: Structure of a Chromosome and Associated Rule

from the set of all the values assumed within the whole training set by the corresponding elements, with the addition of the $-\infty$ and $+\infty$ values. The constraint to be observed is that, for each gene, the value of $x_{i_{MIN}}$ cannot be greater than the value of $x_{i_{MAX}}$. After the computation of the fitness value for each generated chromosome, the reproduction process starts. A hybrid method was designed for the selection of the parents. This method can be considered as a binary selection double tournament with steady-state replacement. The algorithm randomly selects two pairs of chromosomes and, within each pair, compares the fitness values. The fittest chromosomes of each selected pair are selected for reproduction and their two children take the place of the losing chromosomes. This technique promotes elitism. In fact, the best chromosomes in each binary tournament are always winning and never substituted. After each step of the reproduction only the new individuals' fitness is recomputed and thereafter they are immediately ready for the reproduction. By using such a mating strategy, it is possible to use a promising individual for mating just as soon as it is created. The used fitness function is:

$$Fitness = \frac{k_1 \cdot \left[\left(\frac{elem}{k_3}\right) + neg + 1\right]}{k_2 \cdot pos + 1} \tag{4.1}$$

where k_1, k_2 and k_3 are three parameters whose optimal values were fixed by an experimental investigation, *pos* and *neg* are respectively the number of feature vectors belonging to the training set which are correctly classified by the rule associated to the chromosome, and the number of feature vectors belonging to the same set which are misclassified, *elem* is the number of elements of a feature vector which are not generalized to the value ANY. We assume that an element is generalized to the value ANY when the corresponding gene covers the whole set of real numbers, that is, when $xiMIN = -\infty$ and $xiMAX = +\infty$. Lower fitness values correspond to better chromosomes; therefore, in the comparison between two chromosomes, the one with the lower fitness is chosen. We insert the number of elements whose corresponding gene is not generalized to the value ANY in the fitness function in order to favor less complicated rules. Having fixed the value of all the other parameters, in fact, a rule with a simpler structure is associated to a chromosome characterized by a smaller value of the *elem* parameter. To obtain a behavior as independent as possible from the used training set, we have adopted an uniform crossover strategy, described in figure 4.3. At

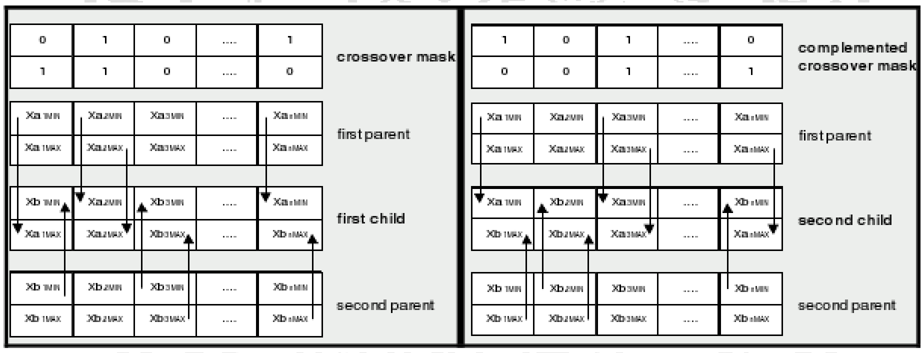


Figure 4.3: The Crossover Mechanism

each reproduction step, a mask made by a pair of values for each gene of the chromosome is randomly generated. Each element of the mask contains the value 1 or 0. When a mask element is 1, the corresponding gene in the first chromosome is selected, and copied in place of the same gene of the first descendant. If it is 0, the gene from the second chromosome is selected. The second crossover is carried out using a mask obtained by complementing the previous one. In other words, the second descendant will have all the remaining genes. It must be guaranteed that the condition $x_{i_{MIN}} \leq x_{i_{MAX}}$ still remains valid after the crossover phase. At the end of the crossover, the new chromosomes undergo the mutation process. A mutation mechanism with incremental probability was proposed, in order to fully exploit the peculiarity of both crossover and mutation. We initialize the mutation probability to a very low value, then we progressively increase it during the genetic analysis. This technique offers the advantage of fully exploiting the mutation capacity of moving a suboptimal far from local maxima. Working as a perturbation, mutation avoids the problems of a slow convergence and, even though in a smaller measure, of a premature convergence. Once the convergence has been reached, the obtained chromosomes are chosen and translated into a rule. If the rule associated to such chromosomes is able to correctly classify all the training set data, the process ends, otherwise it restarts and tries to identify a new chromosome able to classify the feature vectors not covered by the previously selected chromosome. At the end of the process, the set of rules that describes the whole decision region is composed by all the rules corresponding to the selected chromosomes.

4.1.2 Experimental Results

The proposed system has been tested on a subset of the database created by DARPA in the framework of the 1998 Intrusion Detection Evaluation Program. It is made up of a large number of network connections related to normal and malicious traffic. This database was pre-processed at the Columbia University giving rise to a feature vector of 41 elements for each connection, according to the set of features defined in [64] and tailored for the intrusion detection problem. Each connection in the database is labelled as belonging to normal traffic or to an attack. It is worth noticing that the attack class is made up of different variants, each one exploiting different vulnerabilities of a computer network. The 1999 KDD intrusion detection contest used a particular version of this dataset. Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it was a true Air Force environment, and also injected multiple attacks in it. Even if this database has been collected in 1998, and some criticisms have been expressed on it [75], it is still widely used for testing the performance of an IDS [42, 68]. The results obtained by means of the proposed system are reported in terms of

i) the overall error rate on the classified packets

ii) the false alarm rate and the missed detection rate on the classified packets

iii) the reject rate

In the following we present the results obtained by our classification method applied to two different network services (SMTP and HTTP) among those present in the DARPA database. The choice of designing a different classifier system for each service follows the so-called *modular approach* presented in [43], where the authors experimentally demonstrate the advantage, in terms of recognition performance, of an IDS that develops a different classification module for each one of the network services to be protected. For each service, a separate feature selection [81] process was performed in order to reduce the data dimensionality. In particular, we have adopted a Sequential Forward Selection strategy, with the Minimum Estimated Probability classification criterion. Moreover, different values of the parameters k_1, k_2 and k_3 (see equation 4.1 on page 98) have been tested. The selected value has been chosen in order to optimize the results on the training set.

SMTP Service

In this case the training data was made up of 9723 patterns related to different attack variants and to the normal class. The Test Set (TS) for this service is made up of 3261 patterns with 3207 normal packets and 54 attack packets. After the feature selection process, each packet was described by a 6-dimensional feature vector. In particular, `duration`, `flag`, `src_bytes`, `hot`, `count` and `dst_src_host_same_src` features were selected (see [64] for

Overall error	False Alarm rate	Missed Detection rate	Reject rate
0.00%	0.00%	0.00%	9.12%

Table 4.1: Results obtained by the proposed system on the TS for the SMTP service

Overall error	False Alarm rate	Missed Detection rate	Reject rate
0.08%	0.08%	0.06%	9.67%

Table 4.2: Results obtained by the proposed system on the TS for the HTTP service

details concerning their meanings). Table 4.1 shows the results achieved by the proposed system on the raining set. These results have been obtained by averaging ten different trials of the genetic algorithm for generating the two sets of rules. As it is evident, we reach an ideal performance in terms of both missed detections and false alarms on the connections classified by the system. This excellent result is paid with about a 9% of reject rate. Moreover, it is interesting to note that among the classified packets the proposed system is able to correctly detect over the 96% of the attacks to the SMTP service. This descends from the fact that an attack is usually spread over several packets, hence missing or rejecting an attack packet doens't necessarily mean to miss the whole attack it belongs to.

HTTP Service

The training data for this service in the DARPA database is made up of 64292 patterns. However, in [42] it has been demonstrated that a dataset of about 15% of the whole HTTP data is sufficient for training classifiers. Therefore, only 8866 samples have been considered as training data. The test set for this service is made up of 40442 patterns with 1195 attack packets and 39247 normal connections. After the feature selection process each connection was described by a 6-dimensional feature vector, as stated in the previous section. The selected features were the same of the SMTP service. Table 4.2 shows the results achieved by the proposed system on the training set. Also in this case, the results reported here have been averaged on ten different trials of the genetic algorithm for generating the two sets of rules. In this case the system exhibits a quite negligible percentage of false alarms and missed detections. On the other hand, it must be noted that among the classified packets, about the 44% of attacks was detected. In order to make a comparison with other systems, it can be noted that the multi-stage classification system proposed in [2] achieved on the HTTP traffic a slightly higher false alarm rate (as much as 0.09%), while the multiple classifier system proposed in [43] exhibited a false alarm plus missed detection rate of 0.54%. This confirms that our system is able to keep the number of false alarms low.

Key Findings

In this thesis we proposed a genetic-based system for intrusion detection. A genetic algorithm is used for building two rule-based classifiers, a misuse-based one and an anomaly-based one. By suitably combining their opinions about each analyzed network connection, a decision engine improved the ability of the system in avoiding detection errors. The proposed system showed a very encouraging behavior from the detection capability point of view. In particular, in case of the smtp service, we observe an error rate which is equal to 0%. On the other hand, we have a not negligible number of rejected packets. Therefore, as a future development of the proposed architecture, we will work on the analysis of the rejected packets with slower but more accurate algorithms, in order to further improve the detection capability of the system.

4.2 A Behavior Knowledge Space for Network Traffic Classification

Once a suitable labelled training database is available, we can apply supervised classification techniques to detect malicious activities in real network traffic. In the following, we will present a multiple classifier system, which is used to evaluate how effective the training is by means of the automatically labelled database, obtained with the techniques described in chapter 3.

Beside that, the system is also presented as a technique for improving classification reliability and reducing the overall number of errors. Conversely to many multiple classifier systems, in [49] the authors proposed a combining rule that does not require the independence of the base classifiers. The independence assumption, in fact, while on one side assures the best detection improvement, is on the other side rarely verified [60]. Hence, it is often a limiting factor in the practical applicability of the techniques which rely on it. The BKS rule aims, obviously, at combining the decisions coming from multiple different classifiers. The information needed to combine them comes from a *knowledge space*, previously built on the basis of a suitable training set, containing patterns whose class is known *a priori*. Such a space records the behavior of all the classifiers on the patterns contained in such a set, hence it is called a *Behavior Knowledge Space* (BKS), and the combining rule it uses is called the *Behavior Knowledge Space rule*. More in details, if K base classifiers are used, the corresponding Behavior Knowledge Space is a K-dimensional space where each dimension corresponds to the decision of one out of the K base classifiers. Given a pattern x to be assigned to one out of M possible classes, the group of classifiers can in theory provide up to M^K different decisions. By denoting with $D_j(x)$ the decision of the $j-th$ classifier on the input pattern x, the array of decisions $(D_1(x), D_2(x), \dots, D_K(x))$ constitutes one *unit* of the BKS, which is an element of the resulting vector space. Each BKS unit U has a set of M associated counters, e_i, $i = 1\dots, M$, which work as accumulators. our case M is equal to 2, so the number of units is $2K$. The BKS combining rule operates in two phases: a learning phase useful for knowledge modeling, and an operating phase for decision-making.

4.2.1 BKS Training

During the learning phase the BKS look-up table is built-up. During this phase, the $i-th$ accumulator tracks the number of occurrences of an input pattern $(D_1(x), \dots, D_K(x))$ which are associated to the i-th class in the training set. Each element x_{tr} of the training set is classified by all the classifiers and the unit (called *focal unit*) selected by the the K t-uple of decisions $(D_1(x_{tr}), D_2(x_{tr}), \ldots, D_K(x_{tr}))$ of the base classifiers is activated. Let us denote this unit with $FU(x_{tr})$. Let's assume that the actual class $C(x_{tr})$ for the input pattern x_{tr} is j; by adding one to the value of e_j, $FU(x_{tr})$ records that one more input pattern belonging to the training set, and associated with class j, has selected it as a focal unit. At the end of this phase, it is possible to calculate the best representative class associated to each unit, say $C(U)$. To define what *best* means in this context, some criteria are available; we chose to define the best representative class for each focal unit as the class that exhibits the highest value of e_i, according to [49], i.e.:

$$C(U) = j \qquad \text{where } j = argmax_i\, e_i \tag{4.2}$$

In other words, the class assigned to each focal unit corresponds to the one which was most frequently assigned to it according to the patterns of the training set.

4.2.2 BKS Operating Phase

In the operating mode, for each pattern x_{test} to be classified, the decisions $(D_1(x_{test}), D_2(x_{test}), \ldots, D_K(x_{test}))$ of the classifiers are collected and the corresponding focal unit $FU(x_{test})$ is selected. Then the class attributed to x_{test} is the best representative class associated to such focal unit, according to the criterion defined above, i.e.:

$$C(x_{test}) = C(U) \qquad \text{where } U = FU(x_{test}) \tag{4.3}$$

Therefore:

$$C(x_{test}) = C(FU(x_{test})) \tag{4.4}$$

A value can be calculated for the reliability of each response from the BKS; in the following we will call such a value $R(U)$. According to [49], where the BKS was presented:

$$R(U) = \begin{cases} \dfrac{e_j}{e_k} & \text{if } e_j > 0 \\ 0 & \text{if } e_j = 0 \end{cases} \tag{4.5}$$

where $k = argmax_{i \neq j}\ e_i$. In other words, $R(U)$ is the ratio between the values associated to the first and the second most representative class of this unit. If the value associated to the most representative class of a unit is zero (i.e., the considered unit was never activated by the patterns belonging to the training set), the reliability of this unit is set to zero. The value of $R(U)$ can be profitably used for choosing to reject a pattern instead of running the risk of misclassifying it. It is worth noting, however, that other methods [112] for evaluating the reliability of decisions coming from a BKS have been proposed, based on the use of the confidence intervals (CI) [55]. Confidence intervals, when used in classification problems dealing with more than one class, might help in deciding whether a class is more representative than another class for a sample under exam, and how suitable the most representative class is for representing the sample under exam with respect to other classes. Generally speaking, CIs give an estimate of how many samples of a distribution are likely to fall in an interval surrounding a particular value of a parameter. In our case, in order to obtain a reliable classification, confidence intervals related to the two most likely classes might not overlap. Therefore, we introduce a parameter which takes into account the distance between the boundaries of the confidence intervals related to the two most likely classes, once a given focal unit U is activated. Such a distance is indicated with Δ_U and is evaluated by the formula (see also [55] for further details):

$$\Delta_U = P_j - \left(P_k + 1.96 \left(\sqrt{\frac{P_j(1-P_j)}{N_U}} + \sqrt{\frac{P_k(1-P_k)}{N_U}} \right) \right) \tag{4.6}$$

where P_i is a measure of the probability that the actual class is i, and is evaluated as:

$$P_i = \frac{e_i}{\sum_{l=1}^{M} e_l}$$

and N_U is the total number of patterns associated to a specific BKS unit U, i.e.

$$N_U = \sum_{l=1}^{M} e_l.$$

The value 1.96 is related to the 95% CI, and is an easy and reasonable approximation when $N > 100$ [55]. Note that the value of Δ_U can be also negative. Hence, the expression for the reliability function we use in the following is:

$$R(U) = \begin{cases} \Delta_U & \text{if } \Delta_U \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

4.2.3 From BKS to t-BKS

Since in the analyzed domain the temporal sequence of the patterns to be classified assumes a particular significance, the BKS can be improved by the introduction of a temporal dimension. In such a case, the number of units becomes $M^{K \cdot t}$, where t is the size of the considered temporal window. Each unit, in fact, has to record a sequence of t outcomes for each of the K classifiers. Hence, the units in the new behavior knowledge space will be represented by arrays of size $K \cdot t$. In operating mode, t successive decisions for each classifier (relative to a sequence of t consecutive patterns) need to be collected. Then, such $K \cdot t$ values will select a corresponding focal unit in the BKS, whose best representative class will be associated to the pattern under exam. The next pattern will be classified by shifting the temporal window one sample forward, thus individuating a (possibly) different focal unit. It is well known that a BKS based approach might suffer from the curse of dimensionality of each unit, as the dimension of the BKS itself grows exponentially with the unit size. Indeed, in our case, we observed that small values of t are enough to improve the detection performance.

4.2.4 Algorithms for Improving Classification

By observing the proposed approach based on temporal information, it is evident that the temporal window of size t does not only contain information about the last t events occurred on the network, but also about all the temporal windows of size $t-1, t-2, \ldots, 1$, the value 1 corresponding to the

original BKS proposal. Obviously, if $t = 1$, only decisions regarding the current packet are taken into account. The choice for the value of the parameter t is carried out according to the results obtained by analyzing the patterns in the training set. In order to choose the optimal value of the temporal window, we use a training-test set different from the one used for building the BKS lookup table. By running several tests, we choose the value of t which allows the system to perform better on such a set according to a minimum error criterion (*min-err*). According to the such a criterion, t is chosen as the width of the temporal window generating the minimum number of errors on the packets contained in the training-test set. The system proposed in [26], which only uses the basic t-BKS idea, presents quite a high rejection rate, though usually outperforming the best base classifier. Hence, we concentrate on developing some algorithms aimed at exploiting temporal information at its best and trying to reduce the number of rejections. Furthermore, by only considering past base classifiers decisions, in case an attack packet is misclassified, it is completely lost and the error is unrecoverable. A causal system does not help in solving this problem; yet, anticausal analysis might work: even in case of an attack packet misclassification, by observing the *future* decisions of the base classifiers, it might be possible to correct the mistake and detect the attack packets correctly. Thus, we decided to implement also an anticausal algorithm, which not only takes past packets into account, but also future packets.
In the following we will describe all the proposed algorithms.

Fixed Window Algorithm

When using the fixed window algorithm sketched in Figure 4.4 on the next page, the value of t is chosen during the training phase in order to minimize the number of errors on the training-test set. The size of the temporal window remains fixed throughout the operating phase. The sample to be classified corresponds to the newest packet in the sequence. Let K be the number of base classifiers; each unit of the BKS will be an array of $K \cdot t$ elements.

Only exact matches between the units found in the training set and those associated to each sequence of packets in the test set allow the system to classify such packets. In case no matching unit is found among the ones derived from the training set, the packet is rejected, and eventually logged for further analysis.

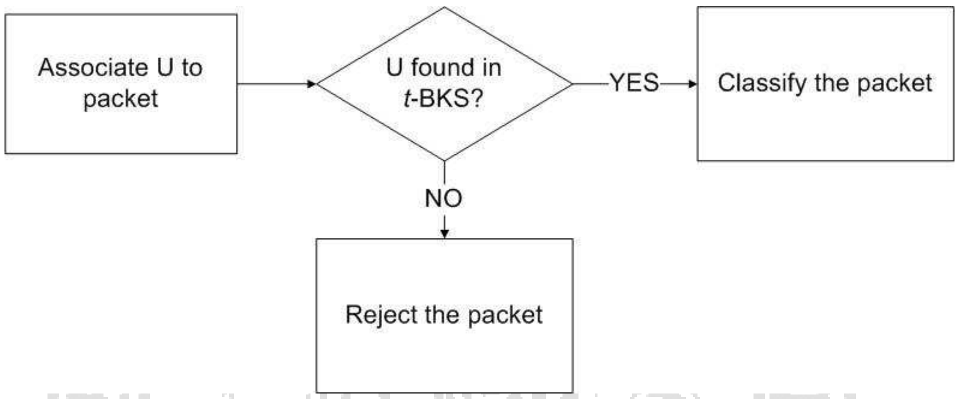


Figure 4.4: Fixed Window Algorithm

Biggest Matching Window Algorithm

Such an algorithm tries to exploit the whole information contained in a temporal window of size t. Once the optimal value for t is chosen during the training-test phase, according to the *min-err* strategy discussed in section 4.2.4 on page 107, a further effort is made with respect to the *Fixed Window* algorithm, in order to find a matching unit in the BKS (see Fig. 4.5). Assuming that we want to classify the most recent of the last t packets, t different logical BKS are built, each related to a size of the temporal window belonging to the set $1, \ldots, t$. This is attained by deleting, at each step, the oldest sample of the sequence. If no match is found within the t-BKS during the operating phase, a new unit of length $(t-1) \cdot K$ is computed, and compared to the units present in the $(t-1)$-BKS. This procedure is iterated until a match is found or until the size of the temporal window reaches the lower bound of 2 without finding any matching unit. In such a case, the packet is rejected.

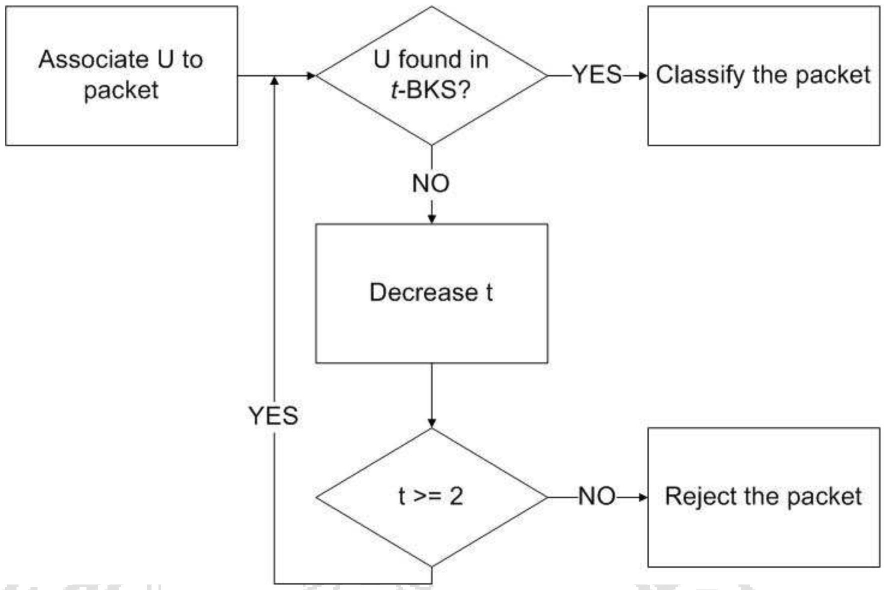


Figure 4.5: Biggest Matching Window Algorithm

Anti-Causal Algorithm

The *Fixed Window* algorithm is used for trying to classify a packet based on the past events occurred. In case no matches were found, the packet is not immediately rejected; instead, the *Biggest Matching Window* algorithm is applied to a window of future events, thus making the analysis anti-causal and introducing a delay in the delivery of the results. Indeed, in case of non strict-real time analysis, if a suitably small size for the future window is chosen, the drawbacks associated with the delay introduced by non-causality would be tolerable with respect to the benefits obtained, in terms of less rejected packets. The optimal values for the width of the past and future window are once again chosen by analyzing the patterns in the training-test set.

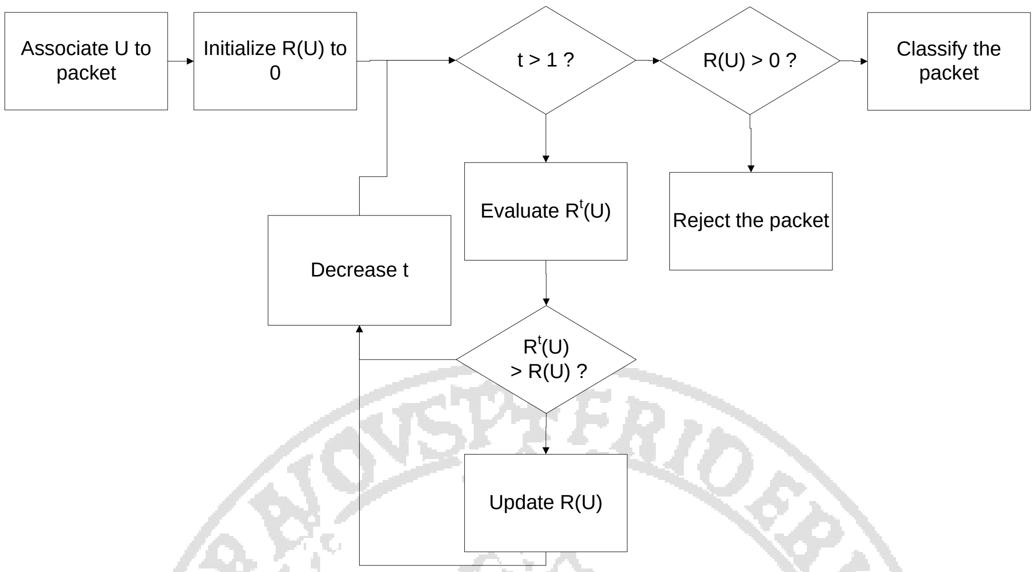


Figure 4.6: Exhaustive Search Algorithm

Exhaustive Search Algorithm

This algorithm aims not only at finding a suitable size t for the temporal window, but also at finding the best value for t related to each sample, according to a specified optimality criterion. In this section we will describe the details of this algorithms, referring to the Confidence Interval (CI) evaluation method described by equation 4.7 on page 107 in order to evaluate the reliability of each classification. The maximum value for t is fixed before the operating phase of the algorithm. Once a unit is associated to the current packet, it is checked against the whole set of $i - BKS$, $i \in \{1, 2, \dots, t\}$. At each step we evaluate the CI associated to the first and the second most representative class in the $i - BKS$ for the examined unit, and assume it is a measure of the reliability of the classification. Hence, we evaluate $R(U)$ (see eq. 4.7 on page 107) and associate such a value to the reliability of the classification. In fact, the more the intervals associated to the first and the second most representative class for a particular pattern are far apart, the more the classification is reliable. When using this algorithm, we need more information with respect to the previous cases. Before the operating phase, in fact, all the $i - BKS$, $i \in \{1, 2, \dots, t\}$ must be always available. When the analysis of each pattern begins, the value of the reliability function is initialized to zero; if it is still zero after all the sizes of the time window have been considered, this reflects the absence of a suitable training pattern for classification. In such a case, the packet is rejected (see Fig. 4.6). The reliability value is only updated in case of positive $R(U)$. Such an algorithms takes into account both the presence of suitable training patterns for classifying each test pattern, and the classification reliability for choosing whether to classify or to reject.

4.2.5 BKS and t-BKS Experimental Results

At the moment, we are running the initial tests aimed at proving the effectiveness of the proposed system in detecting intrusions and rejecting as few packets as possible. Our experimental setup consists of two base classifiers, namely Slipper [23], a rule based classifier based on boosting, and a Learning Vector Quantization classifier.
The tests were performed on a dataset provided by the laboratories of the Italian National Research Council (CNR[1]) [34]. We split the whole dataset, which consists of approximatively 10^6 packets, into three parts, named Set_0, Set_1, and Set_2, respectively corresponding to $30\%, 30\%$ and 40% of the whole data set. The training and training-test set are chosen among Set_0 and Set_1, while Set_2 will be always used as test set. In the following, for the sake of brevity, we will just present the results attained with two values of t, selected according the optimization criteria discussed in section 4.2.4: : we choose, respectively, the value which minimizes the errors, and the value which minimizes the number of rejected packets on the training set. In general, t ranges from 2 to 7, in order to take into account the curse of BKS dimensionality. The presented results were obtained by averaging the performance evaluation attained on the same test set Set_2, related to two different choices of

[1]CNR — Consiglio Nazionale delle Ricerche

the training set to use:

- **Test 1**: Training set $\equiv Set_0$; Training-Test set $\equiv Set_1$
- **Test 2**: Training set $\equiv Set_1$; Training-Test set $\equiv Set_0$

The value of t is fixed during the training phase, by analyzing the patterns contained in the training-test set, according to minimum error criterion. In

Table 4.3: Base Classifiers – Results obtained on the test set

Classifier	**Error Rate**
Slipper	0.2017%
LVQ	0.6406%

Table 4.3 we present the error rate attained by each of the base classifiers. By comparing such results with first row of Table 4.4 on the following page, we observe that the BKS performs at least as well as the best of the base classifiers.

Then, with respect to the results presented in Table 4.4 on the next page, we show how such results change when using the t-BKS. Here, FW stands for *Fixed Window*, AC stands for *Anti Causal*, BMW for *Biggest Matching Window* and ES for *Exhaustive Search*. The numbers in the first column represent the values selected for t on the training-test set for each considered algorithm.

Throughout Table 4.4 on the following page, the error rate always decreases when using any of the proposed algorithms. When analyzing the column reporting the rejection rate, it is noteworthy that in the case of standard BKS we have no rejected packets. This is due to the fact that, in our example, $t = 1$, $M = 2$ and $K = 2$. Hence, there are only four possible units. In such a case, a matching unit is always found during the test phase, but erroneous detection results might be generated. When introducing the temporal dimension, we reduce the a-posteriori probability of error, by making occasional matches with wrong units less likely to happen. Indeed, when using the rejection option as defined in Section 4.2.4 on page 107, some of the errors committed before fall into the number of rejected packets. It means that, when the system isn't confident enough with its detection, it rejects the packet instead of classifying it incorrectly. This is particularly true when the FW algorithm is considered; by rejecting unreliable patterns this algorithm attained the lowest error rate. On the other hand, the AC algorithm is able to reduce the rejection rate with respect to the FW algorithm, almost without affecting the error rate. Finally, it is worth noting that both BMW and ES algorithms make further efforts for classifying packets, hence reducing the rejection rate to 0%. The latter algorithm is the one that attained the best overall performance.

Table 4.4: Multiple Classifier Systems - Results obtained on the test set

Algorithm	**Error Rate**	**Rejection Rate**
BKS	0.2017%	0%
7-BKS/FW	0.1916%	0.2351%
7-BKS/BMW	0.1929%	0%
4-BKS/AC	0.1926%	0.1996%
6-BKS/ES	0.1926%	0%

4.3 An Overlay Network for Cooperative Intrusion Detection

Network services and applications mandatorily impose to adapt the network to new users' needs through the development of innovative techniques for traffic and resource engineering. Given the strict correlation between users' behavior and network resources management, it becomes of paramount importance to obtain users' information in order to optimize network operation. To assure this requirement, we propose a comprehensive approach, named Behavioral Network Engineering, which aims at exploiting data about users' behavior in order to effectively manage, secure and dimension the network. Behavioral Network Engineering exploits information about both the user's behavior and the current network status in order to define the proper actions to be performed onto the network. "Context" information and users' profiles together contribute to the extraction of the behavioral knowledge needed to successfully build an autonomic security system. As stated in chapter 1, network security systems, and in particular Intrusion Detection Systems, represent the main networking application exploiting users' behavioral information in order to detect malicious activities ongoing in the network. The architecture presented here, can be used to deploy multiple classifiers, whose outcomes will be combined in the fashion described so far in this chapter. An IDS needs to know what all users are doing, in order to effectively face the threats carried out by the malicious ones. Moreover, this information must also take into account the relationships among the sets of users sharing common features, resources and purposes. Unfortunately, current real-time intrusion detection systems do not adopt any user characterization including such "inter-users" information. A Distributed Denial of Service attack, for example, is generally realized through coordinating activities involving actors which are widely distributed throughout the network. According to this property, we claim that the detection of such attacks also requires the knowledge of information regarding the "cooperation" among different network entities. Such entities share the same "purpose" (i.e. the compromission of either a service or a single host) and make use of the same resources (e.g. those belonging to the network infrastructure) in order to fulfill their task. For this reason, the set of hosts involved in a DDoS attack can be considered as a "community", whose members all strive for the same result. Thus, the information about this community as a whole can also contribute to realize a deeper knowledge, i.e. the behavioral information, needed to improve the detection process. According to these considerations, for the proposed IDS we have adopted a user behavior model which includes both single user and community information. We propose to look at the network as a self-aware and a self-healing environment. In such an environment, a distributed framework for network protection, still complying with the inherently monolithic IDS structure described in section 2.3.1 on page 35, and capable of providing a previously agreed upon protection level, is well suited. A distributed

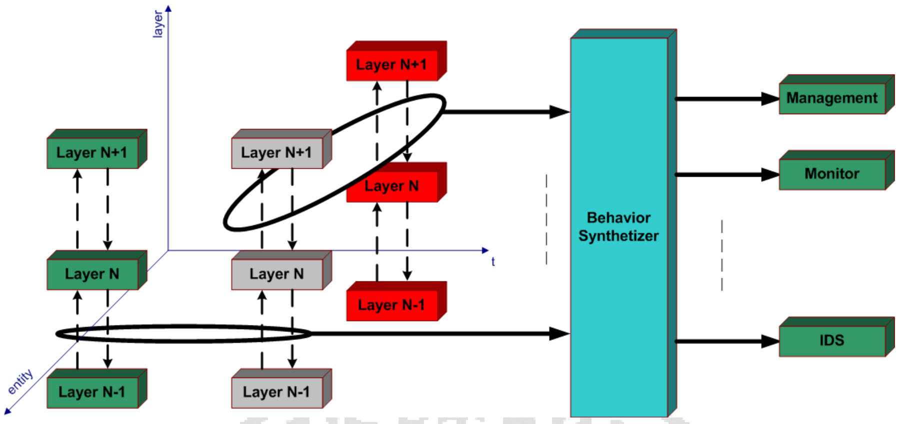


Figure 4.7: Behavioral Network Engineering for Intrusion Detection

system (figure 4.8) allows the separation of concerns among a well-defined set of entities, each suited to deal with a particular aspect of the problem. This on one hand simplifies the task of each involved entity, and on the other

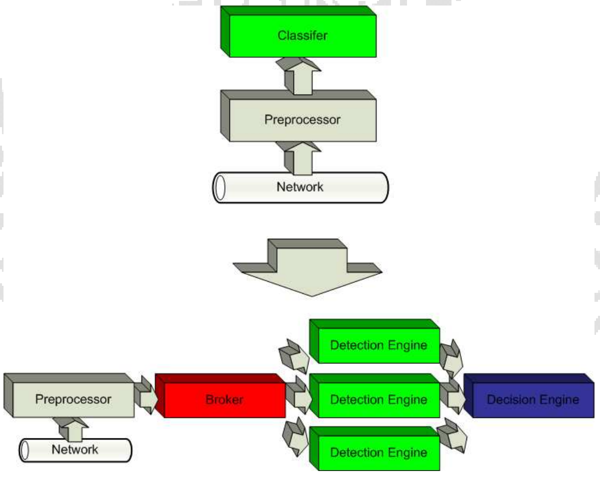


Figure 4.8: Distributed network monitoring for intrusion detection

hand allows a deeper specialization of each module (which can thus be modified without necessarily affecting performance of the overall system). If we assume that the network be aware of itself, security assurance might be regarded as a service inherently provided by network infrastructures. In such a scenario, a framework capable to deploy, both proactively and reactively, on-demand security services is well suited. The IDS structure described so far in this section relies on single entities, each performing just one of the prescribed tasks. A distributed IDS, instead, would naturally adapt itself to a much more context-aware deployment of the available resources, thus allowing a more effective placement of the modules. We propose a system which, by means of a grid-based infrastructure, dynamically deploys the entities in charge of providing network security, based on the knowledge of the security status of the network and its components (figure 4.9). In order to accomplish

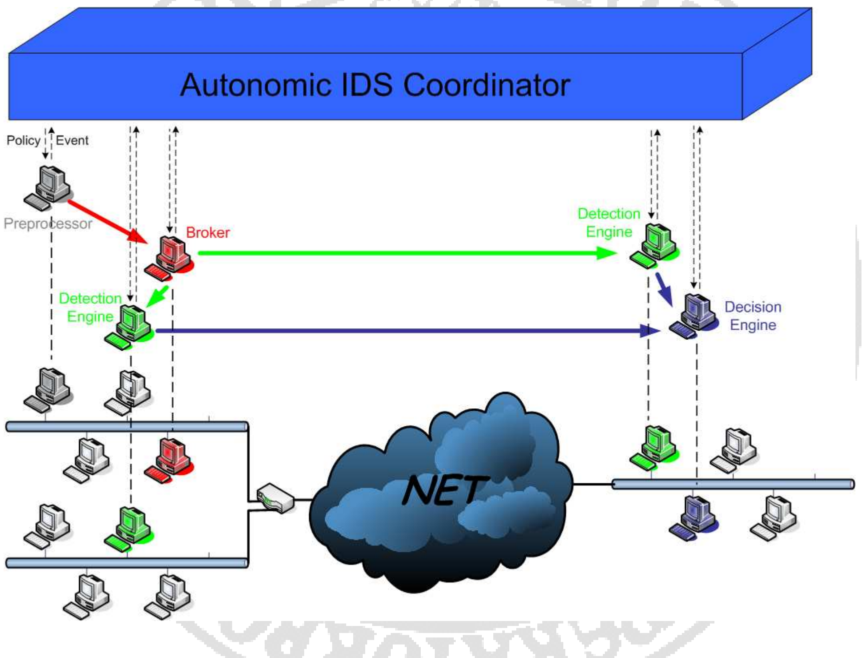


Figure 4.9: Autonomic Intrusion Detection System

its task, the IDS might need several probes sniffing traffic in crucial points of the network, and several classification nodes, each exploiting the best fitting detection technique according to the system status and node location. A broker entity is also needed, capable to instruct all the network nodes to execute a specific application server together with the appropriate grid services. Such a dynamically distributed system might, for instance, adapt itself at the occurrence of a DOS attack, by appropriately placing intrusion detection engines in the most critical nodes (i.e. the nodes along the attackers' path) and by coordinating such nodes through a flexible signalling protocol (e.g. a protocol for tracing back the attack). As far as intrusion detection is concerned in this thesis, there exist four possible outcomes for a system attempting to classify users' behaviors. Indeed, besides the correct detection of normal behavior or of an attack pattern, a classifier might mistake an attack for normal behavior, thus resulting in a missed detection, or some normal operations for an attack, thus generating a false alarm. The correctness of such outcomes of a classifier is the main criterion for evaluating the effectiveness of such instruments. Though mistaking the occurrence of an event for any other event might seem equally harmful, regardless which class of events is mistaken for which other class, not all errors are equally severe. In the context of intrusion detection, missed detections have different importance than false alarms, and tougher endeavors are needed for coping with the minimization of the one out of such two parameters which is deemed to be the most important. Unfortunately, there exists a well known trade off between false alarms and missed detections, which can't thus be minimized together. By exploiting the natural capability of the aforesaid distributed framework, we can try to decrease the number of classification errors by exploiting the detection capabilities of several classifiers of different types, also using artificial intelligence algorithms. While a single classifier can't be too specialized in solving each of the problems it faces, several smaller classifiers may be chosen in order for each one of them to cope with a subset of the problem. This allows us to reach a higher specialization degree, and to perform a better detection on the few attack classes that a single classifier can discover within the network traffic. Multiple classifier systems are widely used in intrusion detection problems, and they usually perform better than systems based on a single classifier, thus encouraging in the exploitation of such a technique.

Chapter 5

Conclusions

Classification theory, and multiple classifier systems, provide a good means for tackling the network security problem. In this chapter we will provide a brief summary of the work behind this thesis, and will draw some general conclusions, by illustrating the key findings. Also, some open issues will be pointed out, leading to directions for future works.

5.1 Our Contribution

The contribution of this thesis are of several types. We've projected systems for the purpose of intrusion detection, by starting from the definition of general architectures. When it was necessary, we extended well known theoretical frameworks, in order to improve and adapt them to the specific application context. Algorithms have been defined and implemented, in order to improve the performance of previously introduced techniques. We developed a comprehensive framework, dealing with most of the aspects related to traffic classification and intrusion detection using supervised and unsupervised classifiers, starting from data collection for the purpose of classifiers' training, and ending with the definition of multiple classifier systems virtually able to operate, by means of proper deployment and organization strategies, in a real network scenario.

First of all, we defined a general reference model which allows us to use classifiers for intrusion detection. The model has been implemented in an architecture, meant to operate in real time, whose performance were tested against variable traffic rates [34]. Such an architecture can be used to extract models, represented as feature vectors, from raw network traffic captured live from the network.

The issue of data collection is challenging from both a scientific and a legal perspective. Several limitations are imposed, by local governments, on the access, the usage and distribution of network traffic traces. However, sharing traffic traces within the research community can help in circulating interesting results, benchmarking novel detection technique, and comparing them against previously proposed traffic classification methods. The privacy issue imposes several limitations on these otherwise useful aspects of common datasets sharing. That's the main reason why we proposed a tool for anonymizing network traffic traces, by carefully removing any sensible information [69]. Raw traffic traces alone, though, are not always of great use for the purpose of classification. When using supervised techniques, or for the aim of benchmarking, all the data contained in the traces has to be correctly labeled, in order to establish the ideal target performance for each classifier. Unfortunately, the operation of labelling is usually long, and requires strong human intervention. We proposed a system based on the Dempster-Shafer combination rule, exploiting multiple different classification techniques, in order to obtained a reliably labeled dataset, in an automated fashion, starting from raw traffic traces, and with virtually very limited human intervention D-S [35]. Due to the inherent uncertainty in labelling raw and unknown traffic traces, we had to extend the formalism of the Dempster-Shafer theory in order to define tools for evaluating the performance, in terms of both accuracy and reliability, of the automated labelling process. The concept of confusion matrix and the metrics used to evaluate error rates had to be redefined, and techniques to evaluate the convergence or the termination of the automated labelling process had to be introduced.

We've also discussed the opportunity to use multiple classifier systems for intrusion detection in real network environments. A simple system based on majority voting has been introduced, combining genetic based classifiers [38]. We gave a definition of the security problem compatible with genetic algorithms, defining chromosomes and mating techniques. We also introduced the rejection option in order to reduce the number of errors. Furthermore, a more complex system based on Behavior Knowledge Spaces has been presented. Such technique suffers from very well know drawbacks, regarding its dimension and its difficulty in training. By suitable choice of the configuration parameters, though, we were able to obtain satisfactory results. Also, we extended the concept of BKS, adapting it to the context of network security. Due to considerations about the high self-correlation in the time domain of attack patterns, we introduced time as a further dimension in the space defined by a BKS, thus defining what we called t-BKS [70]. By analyzing time series of events, we were able to improve the classification performance of the system, though this affected the number of rejected packets, due to increased difficulty in matching long sequences of events. That's why several algorithms were presented and evaluated, which allowed us to reduce the number of rejected packets, without affecting the number of errors. An architecture was presented, which can guide the deployment of multiple classifiers in a network. Such architecture defines, based on certain deployment strategies, a method for collecting information coming from multiple classifiers, and combining such information about users' behavior, in order to obtain a better understanding of the *security status* of the monitored network.

5.2 Key Findings

The core of this thesis consists in the definition and employment of systems based on multiple classifiers for intrusion detection. As to data collection, the problem definition made labelling of traffic really blind. We assumed to have no prior knowledge about the analyzed traffic trace, and aimed at reliably associating labels to each packet. Furthermore, we also aimed at reducing human intervention to the least possible degree. The possibility to use multiple classifiers was somewhat suggested by the process of human learning. When collecting partially complete and partially reliable information from multiple sources, or *teachers*, it is possible, by suitably combining and elaborating it, to build a knowledge whose quality has improved with respect to the one provided by the sources. We had to choose a combination rule which didn't require training, and had no relationships with the data. Also, we had to cope with the limited knowledge about the distribution of classes over the dataset. That's why we needed a technique which allowed us to characterize the base classifiers, rather than the properties of the data. The reliability evaluation mechanism we introduced, proved us that not in every case very reliably labelled, small training sets are better than large but less reliably labelled ones, for supervised classifiers training. Very reliable labels only characterize small portion of the original raw trace, since they are harder to obtain than less reliable labels. Hence, there is an inverse proportionality between reliability and percentage of raw packets that can be labelled reliably enough. Under certain circumstances, the supervised classifiers tested with the automatically labeled datasets performed better when trained on large unreliable datasets, rather than small very reliable ones. This can be summarized by saying that in some cases *quantity is better than quality*. Multiple classifiers worked well also on real traffic classification. Due to different constraints, we were allowed to use combination techniques which require a training phase, such as BKS. In this case, the lack of the independence requirement for the base classifiers really made the task of selecting base classifiers easier. By considering temporal sequences of events for attack detection, we showed how it's possible to improve the detection performance of a common BKS, which respectively performs at least as well as the best of the base classifiers. Though the rejection rate is affected by the higher complexity of time series analysis, by introducing suitable algorithms which explore the time series of events in depth, we were able to compensate such effect, obtaining overall satisfactory results and improvements in performance.

5.3 Open Issues and Future Works

The main objective we had in mind when developing the D-S based system for traffic labeling, was to find a way to automatically do the tedious and difficult job of reliably labelling a huge number of packets. Obviously, our efforts for the future will aim at making the process even more effective, and more reliable. That's why we want to introduce, besides totally supervised and totally unsupervised classifiers, some self-supervised classifiers. So far, we have defined a method for automatically assigning the bpa of a supervised classifier. Such a method relies on the definition of a cost function in the evaluation of the estimated number of errors. A thorough analysis of the effect of different choices for such a function is still missing, and is needed in order to be able to foresee the expected results, given a choice of base classifiers, and of such a function. Also, a more extensive exploration of the available classifiers has to be performed. This could allow us to make choices, based on well defined properties of the base classifiers and on the requirements of the problem at hand. For the problem of multiple classifiers for attack detection in a real network, we proposed some solutions. For them, once again, a theoretical framework for the wise choice of base classifiers is partially missing. In the case of the t-BKS, we will work on the development of a theoretical proof of the properties we evaluated by experiments. The algorithms for reducing the number of rejections showed some interesting properties, which deserve a formal explanation. We will also perform further experiments on the anticausal version of the t-BKS, which could help in recovering from past detection errors. Even more complex algorithms can be implemented, based on refined time series analysis techniques. For example, algorithms such as Viterbi's for Trellis Coded Modulation could be used. Also, the D-S theory used for offline traffic labelling, can be conveniently used for the purpose of traffic classification. It can be interesting to use in case of problem characterized by incomplete specification in terms of probability, which frequently happens in attack detection. Last, but not least, all the presented proposals for solutions have been used in the field of network security. Actually, we proposed them with this specific application field in mind, but they could be adapted to any other field, given a proper problem representation. The definition of features for botnet detection was just an example, but it could be challenging to define and test models for problems typical of completely different areas in classification theory.

Bibliography

[1] Tcpdpriv: Program for eliminating confidential information from traces. WWW. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[2] Network intrusion detection by a multi stage classification system. volume 3077 of *Lecture Notes in Computer Science*, pages 324–333. Springer-Verlag, Berlin, 2004.

[3] E. G. Amoroso. *Fundamentals of computer security technology*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[4] D. Andersson. Detecting usual program behavior using the statistical component of the next-generation intrusion detection expert system (nides). Technical report, Computer Science Laboratory, 1995.

[5] D. Bostock Aristotele, R. Waterfield. *Physics*. Oxford University Press, Oxford, UK, 1996.

[6] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3):186–205, 2000.

[7] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[8] R. G. Bace. *Intrusion Detection*. Macmillan Technical Publishing, January 2000.

[9] D. Barbara, J. Couto, S. Jajodia, L. Popyack, and N. Wu. Adam: Detecting intrusion by data mining. pages 11–16. IEEE, 2001. Workshop on Information Assurance and Security.

[10] R. Barden, D. Borman, and C. Partridge. Rfc 1071 - computing the internet checksum, 1998.

[11] P. Barford and M. Blodgett. Toward botnet mesocosms. In *proceedings of HotBots*, 2007.

[12] P. Barford and V. Yegneswaran. In D. Maughan D. Song M. Christodorescu, S. Jha and C. Wang, editors, *Malware Detection*, volume 27 of *Advances in Information Security*, pages 171–191. Springer US, March 2007.

[13] J. Beale and J. C. Foster. *Snort 2.0 Intrusion Detection*. Syngress Publishing Inc., Rockland, MA, 2003.

[14] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, pages 7–7, Berkeley, CA, USA, 2006. USENIX Association.

[15] M. Bishop. Vulnerabilities analysis. In *proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99)*, 1999.

[16] A. Bloom. *The Republic of Plato*. Basic Books, New York, NY, 2nd edition, 1991.

[17] CERT Coordination Center. Overview of attack trends. Online, February 2002.

[18] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. WWW, 2001. http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[19] T. M. Chen and V. Venkataramanan. Dempster-shafer theory for intrusion detection in ad hoc networks. *IEEE Internet Computing*, pages 35–41, November-December 2005.

[20] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[21] F. B. Cohen. *Protection and security on the information superhighway*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[22] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Machine Learning Conference*. Morgan Kauffman, 1995.

[23] W. W. Cohen and Y. Singer. Simple, fast, and effective rule learner. In *Proceedings of the 16th National Conf. on Artificial Intelligence and 11th Conf. on Innovative Applications of Artificial Intelligence*, pages 335–342, Orlando, FL, USA, July 1999.

[24] D. Conte, P. Foggia, J.-M. Jolion, and M. Vento. A graph-based, multi-resolution algorithm for tracking objects in presence of occlusions. *Pattern Recognition*, 39(4):562–572, 2006.

[25] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association.

[26] L. P. Cordella, I. Finizio, C. Mazzariello, and C. Sansone. Using behavior knowledge space and temporal information for detecting intrusions in computer networks. In S. Singh, M. Singh, C. Apte, and P. Perner, editors, *Pattern Recognition and Image Analysis, Part 2*, volume 2 of *Lecture Notes in Computer Science*, pages 94–102. Springer-Verlag, Berlin, August 2005.

[27] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Learning structural shape descriptions from examples. *Pattern Recognition Letters*, 23(12):1427–1437, 2002.

[28] L. P. Cordella and C. Sansone. A multi-stage classification system for detecting intrusions in computer networks. *Pattern Analysis and Applications*, 10(2):83–100, 2007.

[29] T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[30] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995.

[31] R. P. W. Duin E. Pekalska. *The Dissimilarity Representation for Pattern Recognition: Foundations And Applications (Machine Perception and Artificial Intelligence)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.

[32] C. Elkan. Results of the kdd99 classifier learning. In *ACM SIGKDD Explorations*, pages 63–64, 2000.

[33] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. *A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data.* Applications of Data Mining in Computer Security. Kluwer, 2002.

[34] M. Esposito, C. Mazzariello, F. Oliviero, S. P. Romano, and C. Sansone. Real time detection of novel attack by means of data mining techniques. In C.-S. Chen, J. Filipe, I. Seruca, and J. Cordeiro, editors, *Enterprise Information Systems*, volume VII, pages 197–204. Springer-Verlag, 2006.

[35] C. Sansone F. Gargiulo, C. Mazzariello. A self-training approach for automatically labeling ip traffic traces. In *Book Computer Recognition Systems 2*, volume 45/2008 of *Advances in Soft Computing*. Springer Berlin / Heidelberg, 2008.

[36] X. Fan and M. J. Zuo. Fault diagnosis of machines based on d-s evidence theory. part 2: Application of the improved d-s evidence theory in gearbox fault diagnosis. *Pattern Recogn. Lett.*, 27(5):377–385, 2006.

[37] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[38] I. Finizio, C. Mazzariello, and C. Sansone. Combining genetic-based misuse and anomaly detection for reliably detecting intrusions in computer networks. In *ICIAP*, pages 66–74, 2005.

[39] P. Foggia, C. Sansone, F. Tortorella, and M. Vento. Character recognition by geometrical moments on structural decompositions. In *ICDAR*, pages 6–10, 1997.

[40] P. Foggia, C. Sansone, F. Tortorella, and M. Vento. Combining statistical and structural approaches for handwritten character description. *Image Vision Comput.*, 17(9):701–711, 1999.

[41] A. Fuchseberger. Intrusion detection systems and intrusion prevention systems. *Information Security Technical Report*, 10(3):134–139, 2005.

[42] M. Fugate and J. R. Gattiker. Computer intrusion detection with classification and anomaly detection using svms. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(3):441–458, 2003.

[43] G. Giacinto, F. Roli, and L. Didaci. Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern Recognition Letters*, 24:1795–1803, 2003.

[44] J. Gordon and E. H. Shortliffe. *The Dempster-Shafer Theory of Evidence*, pages 272–292. Addison-Wesley, 1984.

[45] A. Wespi H. Debar, M. Dacier. Towards a taxonomy of intrusion-detection systems. *Comput. Networks*, 31(9):805–822, April 1999.

[46] V. Paxson R. Sommer H. Dreger, A. Feldmann. Operation experience with high-volume network intrusion detection, October 2004.

[47] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[48] J. D. Howard. *An analysis of security incidents on the Internet 1989-1995*. PhD thesis, Pittsburgh, PA, USA, 1998.

[49] Y. S. Huang and C. Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):90–94, January 1995.

[50] N. Cristianini J. Shawe-Taylor. *Support Vector Machines and other kernel-based learning methods*. Camridge University Press, 2000.

[51] C. Tomasi J. Shi. Good features to track. *Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[52] D. Koukis, S. Antonatos, D. Antoniades, P. Trimintzios, and E.P. Markatos. A generic anonymization framework for network traffic. In *Proceedings of ICC 2006, Istanbul, Turkey*, volume 5, pages 2302–2309, June 2006.

[53] I. V. Krsul. *Software vulnerability analysis*. PhD thesis, West Lafayette, IN, USA, 1998. Major Professor-Eugene H. Spafford.

[54] S. Kumar and E. Spafford. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference*, pages 194–204, 1995.

[55] L. I. Kuncheva. Switching between selection and fusion in combining classifiers: An experiment. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 32(2), 2002.

[56] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley Interscience, 2004.

[57] L. I. Kuncheva, J. C. Bezdek, and R. P. W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34:299–314, 2001.

[58] R. Marcondes Cesar Jr. L. da Fontoura Costa. *Shape Analysis and Classification: Theory and Practice*. CRC Press, 2001.

[59] J. Laaksonen E. Oja L. Holmstroem, P. Koistinen. Neural and statistical classifiers taxonomy and two case studies. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 8(1):5–17, 1997.

[60] C. A. Shipp R. P. W. Duin L. I. Kuncheva, C. J. Whitaker. Is independence good for combining classifiers? In *Proceedings of ICPR 2000*. IEEE Press, 2000.

[61] K. Labib and R. Vemuri. Nsom: A real-time network-based intrusion detection system using self-organizing maps. Technical report, Department of Applied Science, University of California, Davis, 2002.

[62] P. Laskov, P. Daussel, C. Schafer, and K. Rieck. Learning intrusion detection: supervised or unsupervised? In *Image Analysis and Processing*.

[63] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[64] W. Lee and S. J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information System Security*, 3(4):227–261, 2000.

[65] J. Leiwo and Y. Zheng. A formal model to aid documenting and harmonizing of information security requirements. In *SEC*, pages 25–38, 1997.

[66] U. Lindqvist and E. Jonsson. How to systematically classify computer security intrusions. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 154, Washington, DC, USA, 1997. IEEE Computer Society.

[67] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[68] Y. Liu, K. Chen, X. Liao, and W. Zhang. A genetic clustering method for intrusion detection. *Pattern Recognition*, 37, 2004.

[69] C. Sansone M. Esposito, C. Mazzariello. A network traffic anonymizer. In *Proceedings of the second Italian workshop on PRIvacy and SEcurity (PRISE 2007), Rome*, pages 4–7, June 6 2007.

[70] F. Oliviero L. Peluso S. P. Romano C. Sansone M. Esposito, C. Mazzariello. Intrusion prevention and reaction: an integrated approach to network security. In L. V. Mancini R. Di Pietro, editor, *Intrusion Detection Systems*. Springer-Verlag, Berlin, 2008.

[71] V. Ciesielski M. K. Ismail. An empirical investigation of the impact of discretization on common data distributions. pages 692–701, 2003.

[72] M. Mahoney. *A Machine Learning Approach to Detecting Attacks by Identifying Anomalies in Network Traffic*. PhD thesis, Florida Institute of Technology, 2003.

[73] T. Mallory and A. Kullberg. Rfc 1441 - incremental updating of the internet checksum, 1990.

[74] G. McDaniel. *IBM dictionary of computing*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[75] 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 System Security*, 3(4):262–294, 2000.

[76] M. Meier, S. Schmerl, and H. Koenig. Improving the efficiency of misuse detection. In K. Julisch and C. Kruegel, editors, *Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Vienna, Austria*, pages 188–205, July 2005.

[77] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[78] J. C. Mogul and M. Arlitt. Sc2d: An alternative to trace anonymization. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data, Pisa, Italy*, pages 323 – 328, 2006.

[79] S. Basu N. Stakhanova and J. Wong. Taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1/2):169–184, 2007.

[80] P. G. Neumann and D. B. Parker. A summary of computer misuse techniques. In *Proceedings of the 12th National Computer Security Conference*, pages 396–407, October 1989.

[81] P. Paclik, R. P. W. Duin, and R. Kohlus G. M. P. van Kempen. On feature selection with measurement cost and grouped features. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *Lecture Notes in Computer Science*, pages 491–515. Springer Berlin, February 2004.

[82] V. Paxson and B. Terney. Bro reference manual, 2004.

[83] L. L. Peterson and B. S. Davie. *Computer networks: a systems approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[84] L. Peluso S. P. Romano G. Ventre R. Canonico, D. Cotroneo. Programming routers to improve network security. In *Proceedings of Workshop On Next Generation Network Programming (OPENSIG2001)*, 24-25 september 2001.

[85] D. A. Frincke R. F. Erbacher, K. L. Walker. Intrusion and misuse detection in large-scale systems. *Computer Graphics and Applications, IEEE*, 22(1):38–47, January/February 2002.

[86] D. G. Stork R. O. Duda, P. E. Hart. *Pattern Classification*. Wiley Interscience, 2nd edition, November 2000. ISBN: 978-0-471-05669-0.

[87] M. A. Rajab, F. Monrose, and A. Terzis. On the effectiveness of distributed worm monitoring. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, pages 15–15, Berkeley, CA, USA, 2005. USENIX Association.

[88] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52, New York, NY, USA, 2006. ACM.

[89] T. R. Reed and J. M. Hans du Buf. A review of recent texture segmentation and feature extraction techniques. *CVGIP: Image Underst.*, 57(3):359–372, 1993.

[90] J. F. Reid and W. J. Caelli. Drm, trusted computing and operating system architecture.

[91] P. Ren, J. Kristoff, and B. Gooch. Visualizing dns traffic. In *VizSEC '06: Proceedings of the 3rd international workshop on Visualization for computer security*, pages 23–30, New York, NY, USA, 2006. ACM.

[92] A. Rijsinghani. Rfc 1624 - computation of the internet checksum via incremental update, 1994.

[93] S. T. Ross. *UNIX System Security Tools*. McGraw-Hill, 1999.

[94] A. Schwartz S. Garfinkel, G. Spafford. O'Reilly, 2003.

[95] M. De Santo, G. Percannella, C. Sansone, and M. Vento. Segmentation of news videos based on audio-video information. *Pattern Anal. Appl.*, 10(2):135–145, 2007.

[96] B. Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[97] B. Schneier. Attack trends: 2004 and 2005. *Queue*, 3(5):52–53, 2005.

[98] S. Singh and M. Markou. Novelty detection: a review - part 2: neural network based approaches. *Signal Processing*, 83(12), 2003.

[99] A. Slagell, Y. Li, and K. Luo. Sharing network logs for computer forensics: A new tool for the anonymization of netflow records. Computer Network Forensics Research Workshop, held in conjunction with IEEE SecureComm, September 2005.

[100] D. D. Sleator and R. E. Tarjan. Self Adjusting Binary Search Trees. *Journal of the ACM*, 32(3), July 1985.

[101] A. Soule, K. Salamatian, and N. Taft. Combining filtering and statistical methods for anomaly detection. In *IMC'05: Proceedings of the Internet Measurement Conference 2005 on Internet Measurement Conference*, pages 31–31, Berkeley, CA, USA, 2005. USENIX Association.

[102] S. Specht and R. Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS 04)*, pages 543–550. ACTA Press, 2004.

[103] W. Stallings. *Network and internetwork security: principles and practice*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[104] R. M. Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Gnu Press, 2002.

[105] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association.

[106] W. Timothy Strayer, C. E. Jones, and B. I. Schwartz. Architecture for multi-stage network attack traceback. In *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 776–785, Washington, DC, USA, 2005. IEEE Computer Society.

[107] B. K. Sy. Signature-based approach for intrusion detection. In P. Perner and A. Imiya, editors, *Proceedings of the 4th international conference on machine learning and data mining in pattern recognition, Leipzig*, volume 3587 of *Lecture Notes in Artificial Intelligence*, pages 526–536, July 9-11 2005.

[108] O. Trier, A. Jain, and T. Taxt. Feature extraction methods for character recognition - a survey, 1996.

[109] M. Tyson. Derbi: Diagnosys explanation and recovery from computer break-ins. Technical report, 2000.

[110] G. Vigna and R. Kemmerer. Netstat: a network based intrusion detection system. *Journal of Computer Security*, 7(1), 1999.

[111] K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In M. Almgren E. Jonsson, A. Valdes A, editor, *Proceedings of RAID 2004*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222. Springer Verlag, Berlin, 2004.

[112] K.-D. Wernecke. A coupling procedure for the discrimination of mixed data. *Biometrics*, 48:497–506, June 1992.

[113] J. Wilander and M. Kamkar. A comparison of publicly available tools for static intrusion prevention. In *Proceedings of the 7th Nordic Workshop on Secure IT Systems*, pages 68–84, Karlstad, Sweden, November 2002.

[114] S. Williams. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[115] L. Xu, A. Krzyzak, and C. Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems Man and Cybernetics*, 1992.

[116] Y. Yang Y. M. Chen. Policy management for network-based intrusion detection and prevention. In *Network Operations and Management Symposium, 2004. NOMS 2004*, volume 2, pages 219–232. IEEE/IFIP, April 2004.

[117] D. Yu and D. Frincke. Alert confidence fusion in intrusion detection systems with extended dempster-shafer theory. In *Proceedings of the 43rd Annual ACM Southeast Conference, Kennesaw, GA*, March 2005.

[118] S. Zanero. Analyzing tcp traffic patterns using self organizing maps. In F. Roli and S. Vitulano, editors, *Image Analysis and Processing*, volume 3617 of *Lecture Notes in Computer Science*, pages 83–90. Springer-Verlag, Berlin, 2005.

[119] C. Zhang, J. Jiang, and M. Kamel. Intrusion detection using hierarchical neural networks. *Pattern Recognition Letters*, 26(6):779–791, 2005.

[120] X. Zhu, T. J. Rogers, R. Q., and Chuck Kalish. Humans perform semi-supervised classification too. In *AAAI*, pages 864–. AAAI Press, 2007.