NTNU – Trondheim
Norwegian University of
Science and Technology

Ensemble-based methods for intrusion detection

Alexandre Balon-Perin

Master of Science in Computer Science
Submission date: July 2012
Supervisor: Bjørn Gamback, IDI
Co-supervisor: Lillian Røstad, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

NTNU – Trondheim
Norwegian University of
Science and Technology

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Ensemble-based methods for intrusion detection

Alexandre Balon-Perin

Supervisor :
Prof. Björn Gambäck
co-Supervisor :
Prof. Lillian Røstad

Master thesis
for the degree of Master
in Computer Science

Academic year 2011 - 2012

Abstract

The master thesis focuses on ensemble approaches applied to intrusion detection systems (IDSs). The ensemble approach is a relatively new trend in artificial intelligence in which several machine learning algorithms are combined. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Moreover, ensembles are particularly useful when a problem can be segmented into subproblems. In this case, each module of the ensemble, which can include one or more algorithms, is assigned to one particular subproblem. Network attacks can be divided into four classes: denial of service, user to root, remote to local and probe. One module of the ensemble designed in this work is itself an ensemble of decision trees and is specialized on the detection of one class of attacks. The inner structure of each module uses bagging techniques to increase the accuracy of the IDS. Experiments showed that IDSs obtain better results when each class of attacks is treated as a separate problem and handled by specialized algorithms. This work have also concluded that these algorithms need to be trained with specific subsets of features selected according to their relevance to the class of attack being detected. The efficiency of ensemble approaches is also highlighted. In all experiments, the ensemble was able to bring down the number of false positives and false negatives. However, we also observed the limitations of the KDD99 dataset. In particular, the distribution of examples of remote to local attacks between the training set and test set made difficult the evaluation of the ensemble for this class of attack.

Keywords: ensemble approaches, intrusion detection systems, artificial intelligence, machine learning, feature selection

Acknowledgements

I would like to express my gratitude to all those who helped me complete this thesis. I am deeply indebted to my supervisors from the Norwegian University of Science and Technology Björn Gambäck and Lillian Røstad for their advice and corrections throughout the writing process of this work. Their expertise in respectively machine learning and software security helped develop my personal knowledge on those subjects.

I also warmly thank Esteban Zymanyi from Université Libre de Bruxelles who despite the distance contributed greatly to this work. I have furthermore to thank Olivier Markowitch and Liran Lerman from Université Libre de Bruxelles for their guidance in refining the topic of the thesis.

Finally, I take this opportunity to thank my family and my friends for their moral support. In particular, my parents who were behind me and inspired me during my entire studies. Their encouragements were always appreciated in difficult times.

Contents

List of Figures

List of Tables

Nomenclature

AIS	Artificial Immune Systems
ANN	Artificial Neural Network
ART	Adaptative Resonance Theory
CERT	Computer Emergency Response Team
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
DR	Detection Rate
DST	Destination
DT	Decision Tree
EC	Evolutionary Computation
FAR	False Alarm Rate
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
GP	Genetic Programming
HIDS	Host Intrusion Detection System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	Intrusion Detection
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
KDD cup	Annual Data Mining and Knowledge Discovery competition

LGP	Linear Genetic Programming
MAC	Media Access Control
MARS	Multivariate Adaptive Regression Splines
ML	Machine Learning
NIDS	Network Intrusion Detection System
NVD	National Vulnerability Database
OSI	Open Systems Interconnections
P	Precision
R	Recall
R2L	Remote to Local attack
RBF	Radial Basis Function
REJ	Rejected (Connection Status)
RST	TCP RST flag
SOM	Self-Organizing Map
SRC	Source
SRV	Service
SVM	Support Vector Machine
SYN	TCP SYN flag
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
U2R	User to Root attack
UDP	User Datagram Protocol

Chapter 1

Introduction

1.1 Context

The Internet has grown tremendously in recent decades. The interconnection of computers and network devices has made the cyberspace so complex that even the best experts on the planet do not fully understand its deepest inner workings. Personal computers get faster every year and it is not rare for a very ordinary person to connect to the Internet through 10 Mb/s lines or faster. To complicate the matter, there is an increase of the number and level of confidentiality of the information found on the Internet. On-line banking and on-line payment make life easier for average people, but make it harder for security experts and network administrators. Practically anyone has access to the huge database of information that is the Internet. In particular, many websites display information about software security and hacking. Others provide hacking toolkits that even the most inexperienced users can launch without difficulty. Add to this the fact that the Internet was not built in the interest of security because nobody could have predicted its dazzling expansion and we have the ingredients of a monumental logic bomb.

In this big disarray, one can imagine how easily a malicious user can perform harmful actions without being threatened or even noticed. To compensate for the lack of security measures in the original Internet protocols, security experts have developed several tools such as anti-virus and firewalls that are now widely used by the majority of users. These tools are designed to protect "normal" users against malevolent ones, called black hat, who exploit vulnerabilities of the system to break into machines containing confidential information or personal data. The motives of these unscrupulous individuals are various: fun, glory, money, identity theft, ideological differences, etc. Their biggest advantages on security experts: they are numerous, they have time and they can attack from anywhere on the planet. In these conditions, passive tools such as firewalls lack the power to counter repetitive and massive attacks of these aggressive hackers.

> "Persistence and determination, combined with plenty of time, gives the attackers the upper hand."
>
> Costin Raiu [61]

Intrusion detection systems (IDSs) are monitoring devices that have been added to the wall of security in order to prevent malicious activity on a system. This work focuses on network intrusion detection systems (NIDSs) mainly because they can detect the widest range of attacks compared to other types of IDSs, as we will see in Section 2.1.3. Network IDSs analyse traffic to detect on-going and incoming attacks on a network. Additionally, they must provide concise but sound reports of attacks in order to facilitate the prevention of future intrusions and to inform the network administrators that the system has been compromised. In the last decade, many scientists and security firms have tried to develop robust IDSs. Unfortunately, most of the time,

they fall short against smart opponents who manage to either evade detection or directly attack the IDS.

Nowadays, commercial IDSs mainly use a database of rules, called signatures, to try to detect attacks on a network or on a host computer. This kind of detection method is currently the most accurate but also the easiest to evade for experienced users because variants of known attacks are considered harmless by the IDS and can pass through without warning. These variants have similar, but slightly different signatures than the original attacks and that is why the IDS cannot detect them. New attacks and attacks exploiting zero-day vulnerabilities can also slip through the security net if their signature is unknown by the IDS. A zero-day vulnerability [26] is a weakness in a software that is unknown by the developers of the vulnerable system and that could potentially allow an attacker to compromise the it. Zero-day refers to the fact that it is the first day or day zero that the vulnerability has been observed in the wild. Since the vulnerability is unknown to the developers of the system, they have not been able to produce a patch preventing exploits to take advantage of this vulnerability.

For these reasons, mechanisms allowing the IDS to learn by itself how to detect previously unseen attacks are needed. Machine learning techniques have come to the rescue, but the challenge is commensurable with the complexity of the problem. An IDS must be able to detect all previously seen and unseen attacks without failure. It must never let an attack pass through unnoticed and it must never deliver unwanted warnings when the traffic is in fact legitimate. These requirements make the task of the machine learning algorithm extremely delicate. In comparison, other fields, in which machine learning techniques has been applied successfully, usually do not require such a high level of precision. This is the reason why machine learning is rarely used in commercial IDSs.

Despite this observation, machine learning experts have been very active in the field of intrusion detection as shown by the numerous papers available on the topic. We will try to summarize the current state-of-the-art in machine learning applied to intrusion detection in Chapter 3, Sections 3.4 and 3.6.2. From these materials, we observe the emergence of a shared conclusion. Machine learning classifiers trigger too many false alarms to be useful in practice. Another problem mentioned by most researchers is the lack of labelled datasets. The only labelled dataset available is the KDD99 dataset which is an adaptation of the DARPA98 dataset created in 1998 by the Darpa Advanced Research Projects Agency (DARPA). This dataset has been severely criticised as we will see in Chapter 4, Section 4.1.

To address these problems, new machine learning paradigms have been introduced in the field of intrusion detection. In recent years, the machine learning community has paid more attention to ensemble approaches. The ensemble approach is a relatively new trend in artificial intelligence in which several machine learning algorithms are combined. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Moreover, ensembles are particularly efficient when a problem can be segmented into subproblems. In this case, each module of the ensemble, which include one or more algorithms cooperating together, is assigned to one particular subproblem. Network attacks can be divided into four classes which are described in Section 2.2: denial of service, user to root, remote to local and probe. Previously, the majority of the solutions proposed by the research community included a single algorithm in charge of detecting all classes of attacks. Instead, in this work, we will specialise one module of the ensemble in the detection of attacks belonging to one particular class.

Another noticeable fact is that each class of attacks is characterized by very specific properties. The differences of the classes can be observed in the particular values of certain variables of their examples in the dataset. However, even though feature selection is often applied in IDSs using machine learning techniques, often only one set of features is selected for all classes of attacks. In this work, one set of features will be selected for each class of attacks according to their relevance to the corresponding class. The next thing to do is to feed the corresponding algorithm(s) with the appropriate set of features. The system can, in theory, reach a very high

accuracy with simulataneous small cost. Moreover, the ensemble can be parallelized by using a multicore architecture. In the best scenario, each algorithm could run on a different core of the processor allowing the IDS to attain extremely high performance.

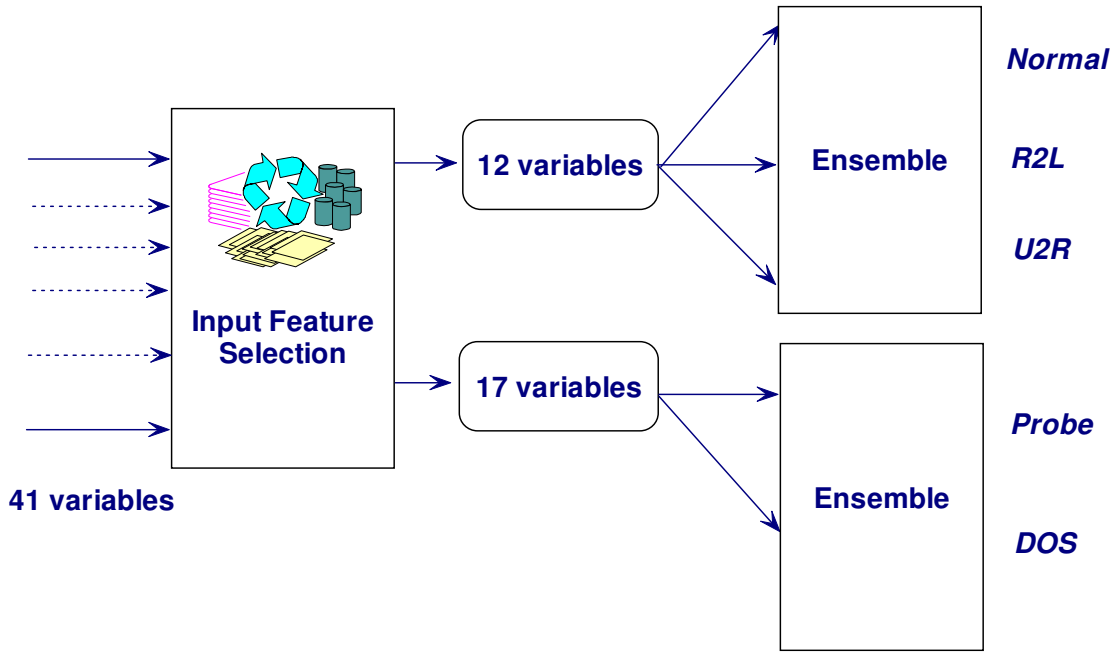


Figure 1.1: Ensemble developed by Abraham and Thomas [6]

Figure 1.1 displays the model developed by Abraham and Thomas [6]. A feature selection algorithm is applied to reduce the number of features of the KDD99 dataset from 41 to 12 features for the three classes normal, remote to local and user to root and 17 features for the classes denial of service and probe. Although their model increases significantly the accuracy of the IDS compared to previously obtained results, there is still much room for improvement. The classes probe and denial of service have very similar properties that is why their features were the same. However, it is natural to expect that there are slight differences even between these two classes. Consequently, by selecting a different set of features for each class of attack, it should be possible to improve these results further. We observed that, generally, 5 or 6 features among the 41 are necessary to characterize one class of attack without losing too much accuracy. Less features means a shorter computation time since less information must be extracted from each packet passing through the network. We will also analyse if a significant drop in the prediction time can be observed when feature selection is applied. In this thesis, we will try to solve the problem of intrusion detection in several steps. The first step will involve an analysis of the most relevant features for each class of attacks as well as an error analysis of the algorithms used with the KDD99 training set. Then, in the second step, the results obtained in the first step will be assessed using the KDD99 test set.

1.2 Contributions

- This thesis provides a survey of the state-of-the-art in the field of ensemble approaches applied to intrusion detection systems.
- Additionally, this work has shown that each class of attacks should be treated separately. In

fact, at least one algorithm should be assigned to detect one class of attacks instead of using a single algorithm to detect all classes of attacks. Furthermore, algorithms used to detect different classes of attacks should be trained with different sets of features.

- The experiments have also concluded that ensembles fed with different sets of features for each class of attacks can outperform more standard approaches even when the ensemble is composed of several simple classifiers such as decision trees.

1.3 Outline of the Thesis

Chapter 2 presents the security aspects of IDSs. More specifically, the main topics of this chapter give an overview of the different types of IDSs, the different methods of detection, the difference between an IDS and an IPS, and the difference between an IDS and a firewall. The requirements to build an efficient IDS as well as the measures used to assess its performance and a possible architecture of a complete functional system are also covered. Finally, the main classes of attacks are described together with examples of attacks in each class.

Chapter 3 focuses on the application of machine learning to IDSs. In particular, the key factors to build an efficient IDS as well as the major challenges in applying machine learning to the problem of intrusion detection will also be summarized. This chapter gives a general overview of the state-of-the-art in the field machine learning applied to IDS and ensemble approaches applied to IDS. Then, the main machine learning algorithms used in the thesis are described. Eventually, the ensemble approach is explained as well as the motives to apply ensembles to the problem of intrusion detection.

The results of the experiments are displayed in Chapter 4. These experiments will try to assess the results of previous works concerning feature selection for ensembles. An ensemble of machine learning algorithms will be fed with the most successful sets of features identified in the previous steps. A description of the main dataset available to train IDSs is also available in this chapter.

Finally, Chapter 5 concludes the thesis and gives a list of possible hints for future work on the topic. Huge challenges await researchers in the field of machine learning applied to intrusion detection. This chapter tries to analyse what could be improved or explored based on the results achieved in this thesis.

Chapter 2

Security

2.1 Intrusion Detection Systems

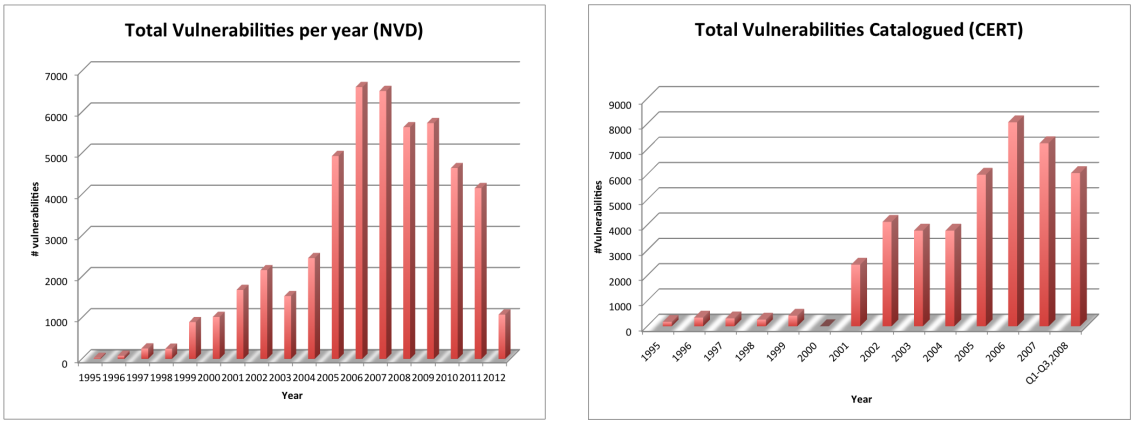
Intrusion detection systems are monitoring devices which are used to detect intrusions on a computer or a network. Intrusions are unauthorized and anomalous activities which were defined by Christopher Kruegel et al. as " a sequence of related actions performed by a malicious adversary that results in the compromise of a target system" [43]. An intrusion detection system is an indispensable tool for network administrators because without such a device, it would be impossible to analyse the huge amount of packets traversing current networks every second. After more than thirty years of intensive research on intrusion detection systems, the field is still open to further investigations especially regarding the accuracy of the detection. Moreover, variants of known attacks as well as new attacks can often go through the system without being detected.

Attacks on a network are possible because software installed on the devices of a network (routers, switches, computers, etc.) contain defects, bugs and flaws. Defects are defined by Gary Mcgraw as "Both implementation vulnerabilities and design vulnerabilities are defects. A defect is a problem that may lie dormant in software for years only to surface in a fielded system with major consequences" [47]. Obviously, the optimal solution to the problem of intrusion would be to have defect-free software. With no software subject to, for example, a buffer overflow attack (Section 2.2.2), detecting it becomes irrelevant. Unfortunately, reality is much less charming because it is extremely difficult to think about every possible misuse of a software when developing it. Moreover, programmers can introduce security breaches when using improperly some programming languages such as `C` and `C++`.

Figure 2.1 shows the evolution of the number of vulnerabilities found in software by year as reported by the National Vulnerability Database (NVD) [56] and the Computer Emergency Response Team (CERT) Cyber Security Engineering team [15]. More than 90% of these vulnerabilities have a risk level ranging from medium to high according to the CVSS Version 2 Metrics [49].

The results obtained by both CERT and NVD are clear: the number of vulnerabilities has increased dramatically in the last decade, but since 2007 the number has decreased steadily. However, more than a thousand vulnerabilities have already been found in the first quarter of 2012 and this number is likely to exceed three thousand by the end of the year. Costin Raiu, director of Kaspersky Lab's Global Research & Analysis Team, summarizes in [61] the major incidents that occurred in the IT Security world during the year 2011. His conclusion is not very encouraging either.

> "If we had to summarize the year in a single word, I think it would have to be 'explosive'."
>
> Costin Raiu [61]



(a) National Vulnerability Database

(b) CERT

Figure 2.1: Total vulnerabilities by year

In particular, Costin Raiu covers the rise of hacktivism that was observed in 2011, major incidents involving worms such as Stuxnet and Duqu, the HBGaryFederal Hack, the Sony PlayStation Network Hack, etc. All these events are worrying because they show that personal information such as credit cards can be easily stolen even when entrusted to big companies like Sony or Google. Furthermore, mobile devices and cloud computing constitute the new nightmare of security experts. Eventually, operating systems such as Mac OS X — previously considered malware-free — are the new targets of cyber-criminals in the last few years. The malware Flashback discovered in September 2011 was probably the first one to break the false sense of security of many Mac users.

With these numbers and incidents in mind, it is easy to understand the urge to improve current software security in general. This thesis proposes to study intrusion detection systems which are one of the key components in modern security along with good cryptography and firewalls. IDSs do not replace current security mechanisms. They are rather a new brick added to the wall of security. Although essential, it is important to remember that IDSs are software subject to bugs and flaws and as such, they can also be targeted by attackers. However, attacks against IDSs is outside the scope of this thesis. For more details about this topic, refer to [50] and [59].

2.1.1 Requirements

When building an IDS, a certain number of requirements must be fulfilled in order for the system to be efficient. Before listing the requirements of an IDS, it is necessary to introduce four important terms.

- **False Positive (FP)**: represents the number of instances that are classified by the IDS as being anomalous when in fact they are legitimate.
- **True Positive (TP)**: represents the number of instances that are classified by the IDS as being anomalous and that really are anomalous.
- **False Negative (FN)**: represents the number of instances that are classified by the IDS as being legitimate when in fact, they are anomalous.
- **True Negative (TN)**: represents the number of instances that are classified by the IDS as being legitimate and that really are legitimate.

FP and FN cause big problem for IDSs. Indeed, an FN represents an attack that was misclassified by the IDS as being a legitimate action. This is the worst possible outcome because it means

that the IDS failed to detect a potentially harmful attack on the network that it was supposed to protect. FP seems to be a smaller problem, but in practice an IDS with a high FP rate will be as useless as an IDS with a high FN rate. For instance, if we consider a 10Gbit/s Ethernet network, the number of packets per second that an IDS should be able to handle vary between 812,740 and 14,880,960 [69]. If the IDS misclassifies one packet every second (or every 14,880,960 packets) as being anomalous when it is not, this means that the network administrator will have to deal with 86,400 false alerts at the end of the day. In this case, the IDS becomes a nuisance for the administrator who will most probably not use it any more even though the primary objective of the IDS was to help him.

An intrusion detection system has to fulfil the following requirements [24].

- **Accuracy:** Also referred as soundness, this property ensures that the IDS does not classify legitimate instances as anomalous. As mentioned above, the problem of false positives limits the use of IDS using anomaly detection (see Section 2.1.6.2) in real-world applications.
- **Performance:** An IDS must be able to classify the traffic without adding a noticeable overload to the network. More details about training and testing times of an IDS are given in Section 2.1.2.
- **Completeness:** This property is the core of the IDS. It states that an IDS should be able to detect all intrusion attempts leading to a false negative rate equal to 0. In practice, this property is very hard to achieve because the IDS must be able to detect known attack as well as unseen ones.
- **Fault Tolerance:** An IDS must itself be resistant to attacks.
- **Scalability:** An IDS must be able to process the traffic of the network in real-time without dropping any packets because of a higher bandwidth than what the IDS can handle. The IDS must be designed in order to be robust in the worst case scenario. For example, in a 10 Gb/s Ethernet network, the largest number of packets that can go through the wire at one moment is 14,880,960. An IDS performing on this type of network should be able to handle that many packets per second. Furthermore, if the IDS uses an audit trail extracted from each host of the network, it must be able to cope with an increase in the number of hosts.

In fact, these requirements are telling us that an IDS must have both a false positive rate and a false negative rate equal to zero. This statement makes it very clear why a good IDS is extremely difficult to build. In particular, applying machine learning to IDS is very arduous in comparison to other applications of machine learning as we will see later in this thesis.

2.1.2 Measures

An IDS's performance is often evaluated by computing measures from the values in the confusion matrix shown in Table 2.1. The confusion matrix shows the distribution of instances that are either correctly classified or wrongly classified by the classifier.

		Actual class	
		Negative class (Normal)	Positive class (Attack)
Predicted class	Negative class (Normal)	True Negative (TN)	False Negative (FN)
	Positive class (Attack)	False Positive (FP)	True Positive (TP)

Table 2.1: Confusion matrix

In particular, the following measures will be used to assess the IDS's performance:

- Accuracy:

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

- False Positive Rate (FPR) or False Alarm Rate (FAR):

$$FPR = \frac{FP}{TN + FP}$$

- Precision:

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

- Recall or True Positive Rate or Detection Rate (DR):

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

- F1score:

$$F1 = 2\frac{R * P}{R + P}$$

- Training time: the time needed by the algorithm to build a model of the data.

- Testing time: the time needed by the classifier (built model) to classify new examples.

The F1score is the most interesting value. It combines the precision and the recall and its range varies from 0 to 1. If the recall or the precision is close to 0 then the F1score will have a value close to 0 as well. In order for the F1score to be close to 1, both the recall and the precision must be high. This ensures that a classifier which, for example, classifies all instances with the same class does not get a high score. In particular, if the dataset is unbalanced, the values of P and R can be misleading. For example, if the dataset contains 1000 negative instances and 500 positive instances, and the classifier always classifies an instance as positive, the recall will be equal to 1 and the precision will be equal to 0.5 leading to a value of F1 of $\frac{2}{3}$. Another popular measure is the FPR which should be as low as possible to avoid unwanted false alarms.

The training time and testing time are also important because they give a good estimate of the performance of the IDS in terms of speed. These values obviously depend on the size of the dataset. In particular, the testing time is highly regarded when working in a real-time environment. The IDS must predict if a packet is anomalous or normal "fast enough" to avoid slowing down the network. "Fast enough" means that it must be able to process a number of packets between 812,740 and 14,880,960 every second when used on a 10 Gbit/s Ethernet network [69]. Figure 2.2 taken from [69] shows the packet rate for different network bandwidths. It is also important to note that additional time should be summed up with the testing time to take into account the time needed to pre-process the data.

2.1.3 Types of IDSs

Intrusion detection systems can be classified into three groups according to their location: host-based IDS, network-based IDS and application-based IDS. These three types of IDSs are briefly described below. For more details about the different types of IDSs, refer to [43].

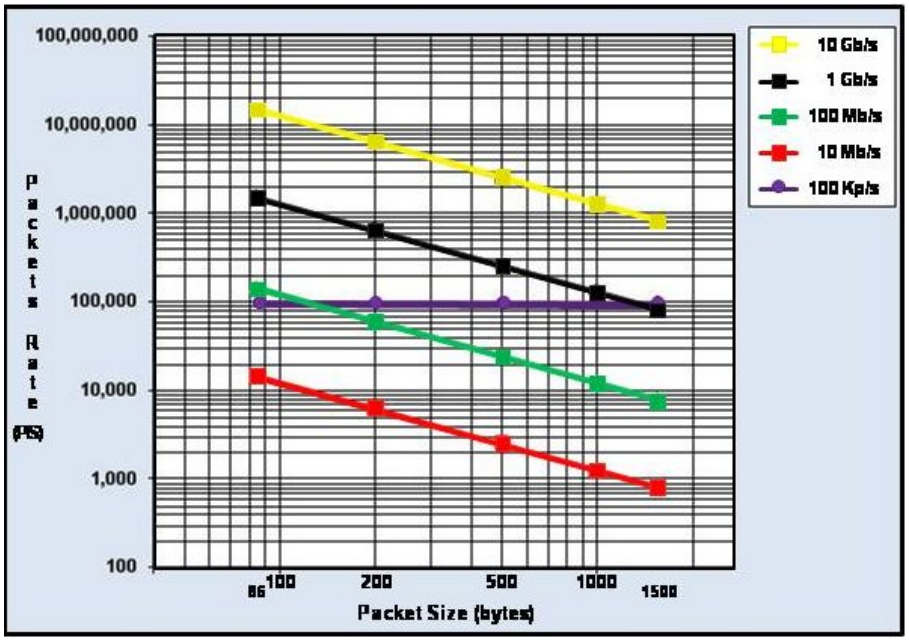


Figure 2.2: Relationship between packet size and packet rate for different network transmission capabilities (from [69])

2.1.3.1 Host-Based IDSs

As the name suggests, an host-based intrusion detection system (HIDS) is located on the host computer. HIDSs analyses audit trail data such as user logs, system calls, etc., on the host where it is installed. Depending on this analysis, the HIDS decides whether an attacker is trying to break into its host or not. There are several advantages of such a type of IDS. First, since the IDS is placed at the end of the communication line, it has access to the full payload of the network packets even if the communication is encrypted. Most of the attacks have valid header and malicious content in the payload of the packet. Therefore, it is very important to analyse the payload in order to detect malicious behaviour. Second, the location of an HIDS facilitates the retrieval of evidences of attacks. Third, it is unlikely that the IDS will miss any packet transmitted to the host where it is installed even if a new route to the network has been found by the attacker.

Nevertheless, an HIDS has also weaknesses. The most important one is that it will be much more difficult for an HIDS to detect a distributed attack on a part of or on the entire network, because it has, normally, only access to content located on its host. Most of the current IDSs do not interact with each other to exchange information on a current attack. Another problem is the scalability. Obviously, large enterprises are the first users of IDSs because they are subject to threats of higher level than smaller sized networks. Having an IDS installed on each system is not the most efficient solution in this case. If the IDS has to be updated or configured, it is much more convenient to have one or a small number of IDSs monitoring the entire network than having one for each host.

2.1.3.2 Network-Based IDSs

Network-based intrusion detection systems (NIDSs) intercept packets passing through a network in order to analyse them and detect possible intrusion attempts. Since networks are dynamic, an attacker could find other routes to access the network, by-passing the IDS layer. In contrast to HIDS, NIDS are unable to analyse encrypted packet payloads except if they have the private keys needed to decrypt them. In this case, the IDS either has to decrypt, analyse and re-encrypt the

packets or it has to decrypt, analyse and send the packets unencrypted to the right destination. In the former scenario, the IDS overloads the network. The latter scenario could lead to security problems for two reasons. First, the dangerous assumption that all hosts on the network are trustworthy is made. Second, the exchanges and storage of the private keys must be handled carefully to keep their secrecy. Another problem arises when network bandwidth is high. In this case, the IDS might have some difficulties to analyse all the packets. The last problem is related to switched networks where the traffic is sent only to the appropriate destination instead of broadcasting it. In that case, the traffic does not necessarily pass through the line monitored by the IDS, making the detection of intrusion rather difficult.

Despite these limitations, this thesis will focus on NIDSs because they are the most convenient type of IDSs when monitoring large networks. In particular, none of the drawbacks described above are insurmountable. For example, before 2001, Cisco had already found a solution to the problem of switched network by incorporating the IDS directly in the switch [38]. Similarly, still before 2001, ISS/Network ICE were capable of sniffing packets on network with bandwidth exceeding the gigabyte [38]. However, it is important to keep these limitations in mind in order to build efficient NIDSs.

2.1.3.3 Application-Based IDSs

Application-based IDSs are located at the application level on a host computer. They can detect intrusion attempts towards a specific application. They are the least common type of IDSs because of their limited scope, but they can be quite important especially if the application to protect is the IDS. An example of application-based IDS is the web-application firewall which will be covered in Section 2.1.5.

2.1.4 IDS vs. IPS

An intrusion prevention system (IPS) is an active IDS which is placed in-line instead of being used as a network tap or a port span. This means that the IPS has the ability not only to detect an attack, but also to prevent it. This prevention can be done by closing the current connection between the attacker and his victim, by reconfiguring the firewall to forbid future connection from the IP address of the attacker, etc. However, automatic reconfigurations can be dangerous. For example, an attacker can deny access to legitimate users of the service protected by the IPS. If the IP address source of the packets containing the attack is replaced by the address of a legitimate user, the IPS will add a new rule to the firewall in order to prevent future connection of this IP address. Also, since the IPS is in-line, an alternative route must be considered in case of failure. Finally, the IPS must be able to handle the bandwidth of the network on which it is placed otherwise the traffic will be slowed down by the IPS.

Timothy Wickham [79] claims that IDSs are dead and should be replaced by intrusion prevention systems. Only producing an alert when an intrusion occurs is definitely out-dated. Nevertheless, the algorithms underlying IDSs and IPSs stay the same and can always be improved. Stating that "Intrusion Detection Is Dead." [79] is somewhat misleading because IPSs rely on IDSs for the detection mechanism. Finally, before starting to use automatic counter-measures, it is important to improve drastically the performance of the IDSs in terms of accuracy and false positive rate. IDSs are still far from living up to the requirements stated above. Clearing this obstacle is obviously a prerequisite to the implementation of IPSs.

2.1.5 IDS vs. Firewall

Basic firewalls [32] are static defence systems that act as filters. They are not capable of recognizing an attack. They usually block all traffic except the packets matching some rules such as packets

destined to a certain port or coming from certain IP addresses. These rules are configured manually by the network administrator according to the network security policy. This means that the efficiency of a firewall depends on how skilled is the administrator. Firewalls can be seen as simple walls that restrict access to a particular location. That is why they must be placed at strategic locations to ensure full efficiency.

In contrast, IDSs — or more specifically NIDSs — monitor traffic in real-time. They analyse each packet passing through the network in order to detect intrusions. They do not block the traffic, but produce alerts when an intrusion occurs or at least when this intrusion is detected. An IDS can check that the rules of the firewall are correct and in some cases, the IDS, called IPS, will be able to reconfigure the firewall to stop future attacks after a detection has taken place. The IDS should be placed behind the firewall to avoid analysing traffic that will be blocked by the firewall anyway.

More advanced firewalls work very similarly to IDS in the sense that they are capable of detecting unknown attacks. For example, web-application firewalls (WAF) [48, 75] such as ModSecurity [10, 71] monitor traffic destined to web servers. The ModSecurity firewall uses a set of rules that must be written by the administrator in order to be efficient in a particular network. Of course, general rules are available out of the box. These rules focus primarily on the Layer 7 of the open systems interconnection (OSI) model where protocols such as Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) perform. In other words, WAFs can be seen as specialised IDSs for web-applications. In fact, the name WAF was only chosen for commercial purposes, as Giovanni Vigna pointed out [77]. They are placed in front of the web server to make sure that all traffic destined to it is intercepted whereas

IDSs have to monitor the traffic on an entire network to detect all types of attacks. This is the reason why they cannot pay as much attention as WAF to the application layer of the OSI model. It is probably a better idea to use a specialised software such as a WAF to protect web servers than using a single IDS which has to detect all kinds of attacks because it provides an additional layer to the wall of defence.

2.1.6 Detection Methods

There are two main methods of detection for IDSs: misuse-based detection and anomaly-based detection. Each method has its advantages and drawbacks and that is the reason why a new category has been studied intensively in the last few years called hybrid detection. The major purpose of hybrid detection is to improve the detection accuracy by combining the strengths of the two main methods previously mentioned: the accuracy of the misuse-based detection on previously known attacks and the generalization power of the anomaly-based detection to previously unseen attacks.

2.1.6.1 Misuse-Based Detection

Misuse-based detection, also called knowledge-based detection, uses signatures of attacks to determine if an attack which is known by the system has occurred. In this type of systems, the IDS has access to a database of signatures which can be local or remote. A signature is a pattern or a sequence of instructions which defines an attack. An IDS using signature-based detection tries to match patterns defined in the list of signatures provided by the signature database and produces an alert if the matching is successful. For example, a signature could contain part of the code of a known virus or a port number which is likely to be used for intrusions.

Using signatures to detect intrusion is a problem for several reasons. Firstly, if the attacker knows what pattern the IDS is looking for, he can easily modify the properties of the packet to bypass the security layer provided by the IDS. Matthew Richard describes a very simple example of encryption which evades the detection of the Snort IDS [63]. Secondly, this dependency on

previously defined models of attacks makes the IDS unable to detect new attacks or variants of known attacks which were not implemented in the signature database [43]. The following appalling assessment reported by Cheng-Yuan Ho et al. [20] in March 2012 shows the inability for misuse-based detection alone to prevail against smart hackers trying to break into private networks.

> "Buffer overflow, SQL server attacks and worm slammer attacks count for 93% of False Negatives even though they are aged attacks because of new variations"
>
> Cheng-Yuan Ho et al. [20]

Basically, IDSs based on signatures are useful when the attacker exploits known vulnerabilities and when the packets cannot be encrypted. For these attacks and only these, misuse-based detection works well because the detection should be 100% accurate and if the signature database is not huge, the detection can be carried out in real time. Even with these drawbacks, misuse detection is still the most widely used detection method in commercial IDSs because it is not prone to false positives. Hence, it is crucial to improve the accuracy of other methods to ensure that variants of known attacks and new attacks can be detected.

2.1.6.2 Anomaly-Based Detection

As described by Abdulrahman Alharby and Hideki Imai [7], possible **anomalous behaviours** that are introduced into the user sessions include, but are is not restricted to logging in from a different source, logging in at an unusual time, executing new commands, and changing identity. The previous list mainly describes anomalies occurring at the host level, unusual packet format and heavy network load are examples of anomalies that can occur at the network level.

Anomaly-based detection systems use collections of data containing examples of "normal" behaviour observed on a network or a host computer in order to build a model of normality for the system being monitored. Any action that deviates from this model is considered anomalous, and therefore, triggers an alert. This kind of detection mechanism allows more flexibility because it is unnecessary to know in advance all types of attacks that can affect the system being protected. In fact, an attack on a network or a host computer usually involves atypical actions which should be flagged by the IDS as anomalous. As a result, anomaly-based detection should normally ensure the detection of new types of attacks or variants of known attacks.

However, in practice, "normal" behaviour is not well-defined and varies from one system to another. Hence, attackers can take advantage of this weakness by disguising their attacks into apparently "normal" actions. Moreover, it is easy to see that this kind of mechanism will trigger many false positives. Indeed, even though the algorithm is capable of generalizing from the training set, any action which was not available in the collection of data used to train the anomaly-based detection system could be considered anomalous. One can easily imagine the huge number of examples that would be needed to cover all possible "normal" behaviour of a user on a system. Obviously, obtaining such a dataset is infeasible. One domain in which anomaly-based detection systems are often used is assembly chains. They must ensure that the product that was created follows the requirements established by the designer. In this case, this kind of detection is very efficient because the characteristics of the wanted object are well-defined.

Finally, for the problem of intrusion detection, it seems that the term "anomaly-based detection" is being used to describe any IDS using machine learning techniques as a detection method. However, machine learning techniques are not limited to anomaly-based detection in which the algorithm is only trained on "normal" examples also called negative examples. Most of the time, machine learning algorithms are trained on datasets containing both "anomalous" (positive) and "normal" (negative) examples.

2.1.7 Architecture

In order to build efficient IDSs, one should think about the different components of their architecture [50]. Generally, IDSs have at least two components: a data collection module and an analysis module.

- **The data collection module** is in charge of collecting pieces of information used as evidence of an intrusion and delivering it to the rest of the system to decide if there is an anomaly. This module collects audit trails such as user logs, network trails, system calls, etc. The data collection module can be seen as a network of sensors monitoring the audit trails in real time.

- **The analysis module** analyses inputs received by the data collection module and decides if there is an intrusion or not. It is the core of the IDS and can vary depending on the detection method. Its purpose is to provide the system administrator with a clear summary of the system's situation. This module can be very resource exhausting. That is why it is often combined with another module which processes the data first to lower the computational complexity as will be seen later (Section 3.2).

More complex IDSs have additional features such as a data preprocessing module, a response module and a signature generator module on top of the two main modules.

- **The data preprocessing module** is in charge of reducing the size of the data in order to improve the computational speed of the IDS. NIDSs have to analyse the traffic in real time without introducing too much delays in the network. For this reason, this module is essential to make the IDS viable in a real environment. More details will be given about feature reduction and feature selection in Section 3.2.

- **The response module** can be either proactive or active. Most of the current IDSs are proactive. This means that they set an alarm when an intrusion takes place, but do not try to counter the attack. Active IDSs (intrusion prevention systems) detect the attack and try to stop it. Different types of reactions are possible. The IDS can close the current TCP connection, reconfigure automatically the firewall to prevent further intrusion, etc.

- **The signature generator module** generates new attack signatures according to the information provided by the analysis module.

One possible architecture for a NIDS is shown in Figure 2.3. The data collection module collects the raw data (log files, network packets, etc.) needed to decide if an intrusion has occurred. The data are then sent to the data preprocessing module which tries to compress them by different means (see Section 3.2) without loosing the information that they contain. The compressed data are sent to the analysis module which is in charge of detecting an attack. The analysis can be conducted in two steps. First, the signature-based detection module is consulted. It uses the entries in the signature database to determine if a known attack has occurred. If that is the case, a signal is sent together with data relevant to the current attack to the response module which has to counter the attack.

If the signature-based detection module does not detect any attack, it hands over the data to the anomaly-based detection module. This module usually runs a machine learning algorithm (see Chapter 3) to determine if the system is under a new attack which is not listed in the signature database yet. If an attack is detected, two simultaneous actions take place. First, the anomaly-based detection module sends appropriate data to the signature generator which creates a new signature based on the data and updates the signature database with the new entry. Second, the anomaly-based detection module sends a signal and data relevant to the attack to the response

module to counter the attack. Optionally, the signature database can be connected to other signature databases throughout the network. In this case, all databases are updated with the new entry generated by the signature generator.

Eventually, if no attack was detected the network packet can go through and reach its destination. Additionally, in case of an encrypted communication, either the data collection module or the data preprocessing module needs a routine to decrypt the data. [28] also explore the possibility to have a distributed architecture for the algorithm itself. In that case, each predictor is trained on a different dataset depending on the location of the computer which hosts it. The trained predictors are then sent to all other nodes of the system over the network and combined together.

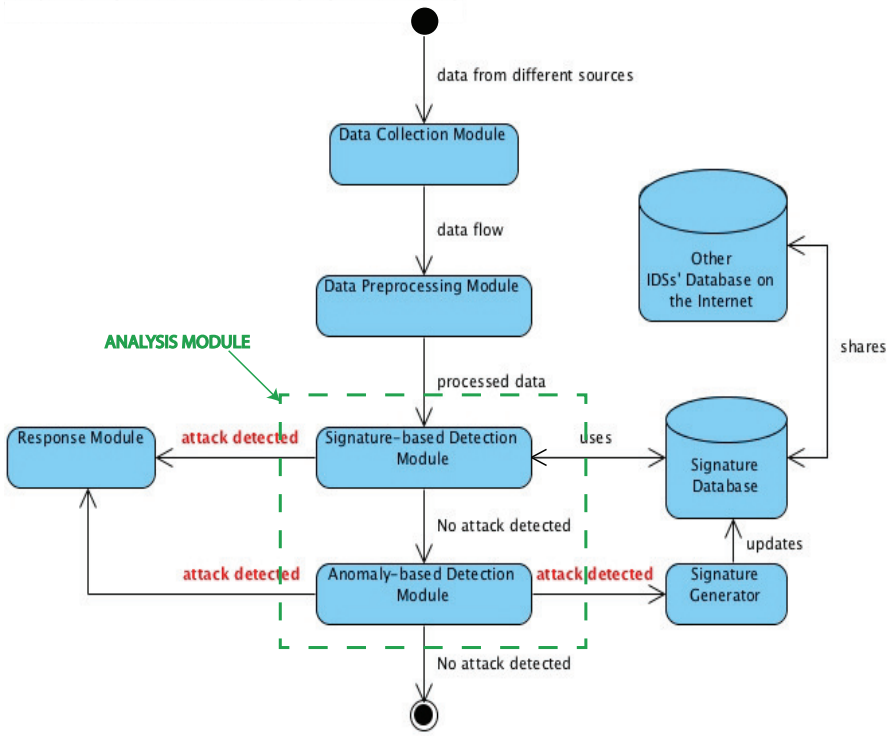


Figure 2.3: Possible IDS architecture

The final aim is to obtain a complete distributed system able to evolve on its own to detect current and new attacks on several networks. Unfortunately, using such a type of architecture would introduce a heavy load in the network because each packet has to go through all modules of the main structure (Data Collection Module, Data Preprocessing Module and Analysis Module). For this reason, each module is generally optimized separately. The entire process described above can be too long to be viable in a real time environment. Especially, the Analysis Module is computationally intensive. The Data Preprocessing Module and the Anomaly-based Detection Module will be the focus of this thesis. In particular, this work will explore the possibility to select features in the data which are relevant to a particular type of attack and which will be fed to a specialised algorithm in order to decrease the computational overload. Obviously, the gain in speed is expected to go without a decrease in accuracy.

2.2 Main Categories of Attacks

A good taxonomy makes it possible to classify individual attacks into groups sharing common properties [50]. One widely used taxonomy divides attacks into four classes: Probes, Denial of

Service (DoS), User to Root (U2R) and Remote to Local (R2L). The class `Normal` that will be used later in this work represents network traffic which is considered attack-free.

Several major concepts must be understood before starting to dig deep into the different classes of attacks.

- **Sniffing [78]:** allows an attacker to take advantage of the fact that many protocols such as FTP or HTTP send information in the clear. Without encryption, an attacker can easily read or "sniff" the content of packets passing through the network. However, sniffing is not always simple. If the attacker is not on the same network or if the network is switched, he must be able to route the traffic through his computer before it reaches its destination. If the traffic is encrypted, the attacker must either break the cipher or steal the keys to decrypt the packets. Sniffing makes it possible for an attacker to steal personal information such as credit card numbers or passwords.
- **Spoofing:** allows an attacker to impersonate another computer by sending packets with a source address different from his computer address. Spoofing is useful to exploit authentication based on the address. In this case, an attacker can access a computer allowing the access to a limited number of addresses.

In general, the best mitigation for all kinds of attacks is to keep a system up-to-date and to avoid misconfiguration. Even though these two recommendations feel as common sense, in practice, many security breaches are due to out-dated systems and misconfiguration. For a more detailed description of each attack and the ways to detect them, refer to the Master Thesis of Kristopher Kendall [39]. The attacks' description can also be found on-line [40].

2.2.1 Probe

`Probe` attacks [39] are often the first step of all other attacks that we have seen previously. They are used to gather information about the targeted network or a specific machine on a network. Without network probes, an attacker would have a hard time finding the vulnerabilities present on his target. That is the reason why it is crucial to detect this type of attacks. However, since probing or scanning abuses a perfectly legitimate feature used by network administrators to check on machines on a network, it is also difficult to differentiate attacks from regular actions. Many programs have been developed to scan a network. The most famous is probably "nmap" which is a powerful tool that can be used to look for active machines and active ports on a machine. This information is very valuable because knowing that the port 80 is active, for instance, means that a web server with potential vulnerabilities or misconfigurations runs on the machine. If port 80 is open, the attacker can also conclude that the machine serves its content unencrypted. "nmap" is not limited to finding the open ports, it is also possible to discover the type and version of the server or the type and version of the operating system. Other attacks such as "saint" and "satan" are specialised in discovering vulnerabilities in the targeted system. These scanning tools allow even unskilled attackers to find vulnerabilities automatically on a large number of machines. A typical attack scenario would involve a first phase where the attacker tries to scan the network that he intends to compromise. Thanks to the scan or probe, the attacker will have a complete map of the machines and services running on the network. The next phase is to find vulnerabilities in the services available with automate programs. Once one or more vulnerabilities are found, the attacker will be able to launch another attack depending on his goals and on the vulnerability found.

Table 2.2, which was adapted from [39], shows different types of `Probe` with some properties for a particular type such as the service that the attack uses, the platforms vulnerable to this kind of `Probe`, the type of vulnerability (mechanism) that the attack takes advantage of, the time required to implement it and the effect caused by the attack.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Ipsweep	icmp	All	Abuse of feature	Short	Find active machines
Mscan	many	All	Abuse of feature	Short	Looks for known vulnerabilities
Nmap	many	All	Abuse of feature	Short	Find active ports on a machine
Saint	many	All	Abuse of feature	Short	Looks for known vulnerabilities
Satan	many	All	Abuse of feature	Short	Looks for known vulnerabilities

Table 2.2: Probe attacks (adapted from [39])

2.2.2 User to Root (U2R)

In a User to Root attack [39], an attacker starts a session on a computer as a normal user with restricted rights and by exploiting some vulnerability on the software installed on the system, the user can elevate his privilege. The goal of this class of exploits is obviously to obtain administrator rights on the attacked computer in order to have full control of it.

There are several different types of `U2R` attacks. Buffer overflow [22] is certainly the major vulnerability used by hackers when trying to obtain privileged rights on a computer. This implementation bug is found mainly in software written in programming languages such as `C` or `C++` which allow the programmer to manually allocate the memory. Memory allocation can be very powerful when used carefully, but is subject to buffer overflow if managed by an inexperienced programmer. The goal of a buffer overflow attack is to corrupt a program running with high privileges (i.e. root) in order to take control of the program. If the program has root privilege, the attacker can immediately execute a command to obtain a root shell. In that case, the attacker has full control of the host computer which runs the vulnerable program. The attack is performed in two steps. In the first step, the hacker must find a way to have the appropriate code to launch a root shell in the memory of the program. To manage that, the attacker uses a buffer with non-existent or poorly performed boundary checking. The second step is to subvert the state of the program. The attacker must corrupt the stack pointer to make it point to his malicious code. Several options are possible but the most common is to overwrite a function return address to point to the first instruction of the code of the attacker. This attack is also called "stack smashing attack" [57]. Other attacks such as "loadmodule" or "perl" take advantage of the way some programs sanitize their environment. Others still ("ps") exploit poor management of temporary files.

Current protections against "buffer overflow" include using automatic detection programs such as `StackGuard` or `StackShield` [21] which check for code pointer integrity, preventing buffers to execute code and performing boundary checking. In some cases, it is possible to bypass these protections (see Phrack magazine [12]). Another drawback is the fact that it is easier to install, update and maintain one or several IDSs to prevent many different attacks on a entire network than a different software for each type of attack on each computer of the network. `U2R` attacks in general can be mitigated by keeping the machines of a network up-to-date and writing correct code.

Table 2.3, which was adapted from [39], shows different types of `U2R` with some properties for a particular type such as the service that the attack uses, the platforms vulnerable to this kind of `U2R`, the type of vulnerability (mechanism) that the attack takes advantage of, the time required to implement it and the effect caused by the attack.

2.2.3 Remote to Local (R2L)

In a Remote to Local attack [39], the attacker starts from a session on a computer outside of the targeted network and exploits a vulnerability in order to gain access to a computer on the

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Eject	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Ffbconfig	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Fdformat	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Loadmodule	Any user session	SunOS	Poor Environment Sanitation	Short	Root Shell
Perl	Any user session	Linux	Poor Environment Sanitation	Short	Root Shell
Ps	Any user session	Solaris	Poor Temp File Management	Short	Root Shell
Xterm	Any user session	Linux	Buffer Overflow	Short	Root Shell

Table 2.3: User to Root attacks (adapted from [39])

local network. A precondition that must be fulfilled is the ability for the attacker to send network packets to the victim host. Very often, but not always, `R2L` attacks are combined with `U2R` attacks allowing the attacker to obtain full access of a remote machine which is part of a different network than the network of the attacker.

Examples of remote to local attacks include "warezmaster" and "warezclient". Those two attacks exploit weaknesses in the file transfer protocol (FTP). The first one grants any user with writing permission on the FTP server. An attacker could use this bug to create a hidden directory and upload illegal files on the server. The "warezclient" attack can be seen as the second step of the "warezmaster" attack since it involves a user downloading the uploaded files from the hidden directory created during the "warezmaster" attack. Other remote to local attacks called "imap", "named" and "sendmail" exploit bugs in well-known protocols used on the Internet such as DNS and SMTP. Attacks exploiting misconfigurations in the system include "dictionary", "ftp-write", "guest" and "Xsnoop". The main mitigation against remote to local attacks is to keep the system up-to-date. These updates will remove from the system the most common bugs that are exploited by `R2L` attacks. Avoiding misconfiguration is a simple, but equally important mitigation method against these types of attacks.

Table 2.4, which was adapted from [39], shows different types of `R2L` with some properties for a particular type such as the service that the attack uses, the platforms vulnerable to this kind of `R2L`, the type of vulnerability (mechanism) that the attack takes advantage of, the time required to implement it and the effect caused by the attack.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Dictionary	telnet, rlogin, pop, imap, ftp	All	Abuse of feature	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	telnet, rlogin	All	Misconfiguration	Short	User-level access
Imap	imap	Linux	Bug	Short	Root Shell
Named	dns	Linux	Bug	Short	Root Shell
Phf	http	All	Bug	Short	Execute commands as user http
Sendmail	smtp	Linux	Bug	Short	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor Keystrokes remotely

Table 2.4: Remote to Local attacks (adapted from [39])

2.2.4 Denial of Service (DoS)

In a denial of service attack [39], an attacker makes a resource on a network either unavailable to legitimate users or too busy or too full to process their queries. The resource can be network bandwidth, computer memory or computing power. There are many different types of DoS attacks.

For example, "ARP poisoning" attack [33, 78] can deny access to a machine on a network. The address resolution protocol (ARP) is a protocol used to convert network layer address (such as the IP address) into link layer address (such as the media access control (MAC) address). Each computer on a network has an ARP table which maps network layer addresses to link layer addresses of the other computers or devices on the network. In an "ARP poisoning" attack, an attacker sends unwanted ARP replies to a user on the same network or replies to an ARP query faster than the destination of the query in order to falsify the information contained in the ARP table of the victim. In this case, it is possible for an attacker to deny access to a resource to one or more users on a network. For instance, depending on the network structure, it is possible for an attacker to modify the entry corresponding to a gateway in the ARP tables of the victim on the network. In this case, the victim might not be able to access the Internet any more. "ARP poisoning" is not represented in the KDD99 dataset, but the concept of denial of service can be easily understood from this attack. It can be noted that "ARP poisoning" can also be used to perform a man-in-the-middle (MITM) attack. A MITM is a type of sniffing attack where the attacker stands in the middle of a communication between two hosts. By poisoning the ARP table of one of the two hosts taking part in the communication, the attacker can redirect the traffic to his computer first and then forward it to the intended destination after having read the content of the message.

The other major type of DoS focuses on resource exhaustion. The attacker sends a huge amount of queries in a short amount of time to the targeted victim. If the victim is a server, resource exhaustion occurs when the server receives more queries than it can process. In that case, legitimate users will not be able to access this resource during the time of the attack or even afterwards if the server crashes. An example of a DoS aiming at exhausting the resource of a machine on a network or an entire network is the "UDP Port DoS" attack, also called "UDP packet storm" [16, 33]. In an "UDP storm", an attacker forges a packet with a spoofed source address of a host running an "echo" or "chargen" process and sends it to another hosts running a similar "echo" or "chargen" process. The receiving host replies with a echo packet to the spoofed source which also replies with another echo packet. A loop is created between the two hosts leading to resource exhaustion or at least, performance degradation. When targeted at a switch or router, the performance of the entire network can be affected.

Another very popular variant of DoS that has been used extensively by hackers in the last decade is the distributed denial of service (DDoS) [45, 17]. A "DDoS" is performed in two main steps. In the first step, an attacker, called master, gains control over a number of computers, called slaves or zombies, by exploiting unpatched vulnerabilities found in the target systems. The number of slaves can vary from one "DDoS" to another but is usually huge, hundreds of thousands of computers is a perfectly reasonable number in most cases. Once the attacker has taken control of a sufficient number of slaves, the second step can start. The master orders all of the slaves to query a designated machine (usually servers) at the same time. The target is flooded with the simultaneous queries. After a short time, the memory of the server is exhausted making it unable to handle all of the queries including the ones from legitimate users. The service proposed by the server is denied.

The mechanism used by DoS attacks can abuse a legitimate feature of a network protocol. Some of these attacks are "mailbomb", "neptune", "smurf" attack and "ARP poisoning". "teardrop" and "ping of death" exploit implementation bugs of the TCP/IP protocol. Finally, attacks such as "apache2", "back" and "syslogd" target a specific program running on the victim host.

Mitigation methods of DoS include disabling unnecessary services such as echo or unused UDP

services, keeping the network devices up-to-date, monitoring the network for anomaly, use proxy mechanisms, avoiding misconfiguration of the firewall and other software used on the network, etc [16].

Table 2.5, which was adapted from [39], shows different types of DoS with some properties for a particular type such as the service that the attack uses, the platforms vulnerable to this kind of DoS, the type of vulnerability (mechanism) that the attack takes advantage of, the time required to implement it and the effect caused by the attack. This list of DoS attacks is by no means exhaustive but gives an overview of the variety found in this class of attacks.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Apache2	http	Any Apache	Abuse	Short	Crash httpd
Back	http type	Any Apache	Abuse/Bug	Short	Slow server response
Land	N/A	SunOS	Bug	Short	Freeze machine
Mailbomb	smtp	All	Abuse	Short	Annoyance
SYN Flood (Neptune)	Any TCP	All	Abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None(*)	Bug	Short	Crash the targeted computer
Process Table	Any TCP	All	Abuse	Moderate	Deny new processes
Smurf	icmp	All	Abuse	Moderate or Long	Network Slowdown
Syslogd	syslog	Solaris	Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Bug	Short	Reboot machine
Udpstorm	echo/chargen	All	Abuse	Short	Network Slowdown
ARP Poisoning	ARP	All	Abuse	Short	Deny access to one or more machines on the network

Table 2.5: Denial of Service attacks (adapted from [39]). (*) the bug has been fixed on most platforms since 1998.

Chapter 3

Machine Learning

Machine learning is a field of artificial intelligence that makes intensive use of statistics. Machine learning algorithms enable computers to make predictions based on a dataset. The set of examples is often symbolised by the capital letter X and represents a matrix of dimension $m \times n$ where m is the number of examples and n is the number of variables in the dataset. Each column represents a variable relevant to the problem being studied. The lines represent an observation or an example of the problem. For example, in the problem of intrusion detection, a column could represent the number of bytes sent from the source to the destination and a line would be a value for this variable (1000 bytes, for instance). One particular instance is written $x^{(i)}$ where i is the index of the line of this instance in the matrix. The entire dataset is written as follows:

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & ... & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & ... & x_n^{(2)} \\ ... & ... & ... & ... \\ x_1^{(m)} & x_2^{(m)} & ... & x_n^{(m)} \end{pmatrix}$$

If supervised learning is used (see Section 3.1), a label is attached to each example of the dataset indicating to which class the example belongs. The set of labels is often represented by the capital letter Y and is a vector of dimension $m \times 1$ where m is still the number of examples in the dataset. One particular label can be written $y^{(i)}$ where i is the index of the example to which the label is attached. The set of labels can be written as follows:

$$Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ ... \\ y^{(m)} \end{pmatrix}$$

There are two types of predictions depending on the type of values that the algorithm must output. The output belongs to the set of all values that the labels can take. Regression analysis is used when the output value is continuous. For example, if the algorithm is used to predict house prices, the output price is a continuous value. The simplest algorithm is the linear regression. Linear regression is simple because it tries to fit a straight line to the data; more complex algorithms such as artificial neural networks build non-linear models. When the predictions are discrete values, the mechanism is called classification. In general, the output of a machine learning algorithm used for classification, also called a classifier, belongs to a small set of discrete values. For example, in the case of breast cancer [53], a binary classifiers classifies examples into two classes depending on whether the cancer is malignant or benign.

Usually, the dataset is divided into a training set, used to train the algorithm and a test set, used to assess the performance of the algorithm on new examples. Generally, a good division of

the dataset includes 70% of the data in the training set and the last 30% in the test set [54]. Another very common division of the dataset includes 90% of the data in the training set and 10% in the test set. In any case, a "good division" depends strongly on the data. Consequently, machine learning algorithms operate in two main steps. In the first step, the algorithm uses the training set to build a model of the data. The model differs greatly depending on the type of algorithm used. Several machine learning algorithms will be covered in Section 3.4. In the case of regression, the algorithm must find the function which fits the data as well as possible. In the case of classification, the algorithm must find decision boundaries that separate the data as well as possible according to the number of desired classes. In both cases, a cost function is used to evaluate how good the model fits the data. The goal of the machine learning algorithm is to find the model that minimizes the cost function.

3.1 Supervised Learning vs. Unsupervised Learning

Machine learning algorithms can be divided into two major classes depending on their learning technique: supervised and unsupervised. Another interesting type of learning method which lays in-between the two main types is called semi-supervised learning because it uses techniques of both supervised and unsupervised learning. This learning paradigm has been studied intensively in the last few years, but not much in the field of IDS.

Supervised learning implies to obtain a training dataset in which every entry is labelled. A label indicates which class the example belongs to. For example, each entry in the KDD99 dataset (described in more details below, see Section 4.1) is originally labelled with the type of attacks it belongs to or `Normal` when the example corresponds to a harmless packet. However, they could also be labelled with only two different labels: "attack" or "normal". Using only two classes allows binary classifiers such as support vector machines (SVM) to learn from the dataset. Another option would be to label the examples according to the class of attack they belong to: `DoS`, `Probe`, `R2L` and `U2R`.

The idea of supervised algorithms is very similar to the process in which a child learns to recognize objects with the help of his parents. One of the parents shows him different objects and pointing at one of them tells the word corresponding to the object. The child keeps track of information such as the shape of the object, its colour, etc. These properties of the object are the variables contained in a dataset. A set of values of these properties represents an example in a dataset. The child then associates a label (the word told by his parent) to the corresponding example in his memory which can be seen as his personal "dataset". The next time that the child will see an object similar to the ones in his "dataset", hopefully, he will be able to utter the word corresponding to the label of the set of examples found. A supervised machine learning algorithm works very similar to this analogy. The algorithm builds a model which should be able to separate the examples with different labels. For example, logistic regression can find decision boundaries to separate the data from different classes. There are many supervised learning algorithms such as artificial neural networks (which can also be unsupervised), support vector machines, decision tree, etc.

The most obvious disadvantage of supervised learning is the need for a labelled dataset. This is a big problem for the IDS research community because the only available dataset with labels is the KDD99 which was created in 1999. Since then, many new attacks have been developed and for this reason mainly, this dataset is considered sometimes obsolete. Nevertheless, it is still widely used because of its uniqueness and because useful information can be extracted from it. In fact, obtaining data is cheap whereas obtaining labels for the data is very expensive both in terms of time and money, because one or more experts must go through millions of examples and assign them a label. Apart from this main drawback, supervised learning has also some advantages. The first one is the ease of use and interpretation of the results. Indeed, the output of the

classifier belongs to one of the classes defined by the labels of the dataset. The second advantage of supervised learning is its accuracy to classify similar examples. However, this accuracy drops significantly when the new examples are not so similar to the ones in the training set [44].

Unsupervised learning algorithms do not need the dataset to be labelled. The most popular technique of unsupervised learning is called clustering. In this case, the algorithm exploits the similarity of the examples in order to form clusters or groups of instances. Examples belonging to the same cluster are assumed to share similar properties and belong to the same class. In contrast to supervised learning, disadvantages of unsupervised learning include the manual choice of the number of cluster that the algorithm must form, the low accuracy of the prediction and the fact that the meaning of each cluster must be interpreted to understand the output. However, unsupervised learning is more robust to big variations than supervised learning. This is a very important advantage that unsupervised learning has over supervised learning for the problem of intrusion detection because it means that unsupervised learning is able to generalize to new types of attacks much better than supervised learning. This property could be very useful to detect zero-day vulnerabilities.

In conclusion, the lack of proper labelled datasets and the speed at which hackers invent new attacks could make unsupervised and semi-supervised learning good candidates for future development of IDSs [44]. In fact, an IDS should be trained with traffic found on the network where it will be deployed in order to take into account the particular configuration of the network. For example, what could be an anomalous behaviour on one network is not necessarily anomalous on another network. In this case, it is obvious that only unsupervised learning can be used to train the IDS. Otherwise, the very expensive task of labelling examples should be performed for each new network needing an IDS. This is evidently infeasible in practice. In this work, supervised learning will be used for its simplicity. This will be sufficient since the point is to show that an ensemble correctly fed with the relevant subsets of features of the dataset will have a higher accuracy and a lower response time than more standard machine learning approaches.

3.2 Feature Selection

Feature selection is a very efficient way to reduce the dimensionality of a problem. Redundant and irrelevant variables are removed from the data before being fed to the machine learning algorithm used as a classifier. Feature selection is a preprocessing step which can be independent of the choice of the learning algorithm or not. It can be used in order to improve the computational speed with minimum reduction of accuracy. Other advantages include noise reduction and robustness against over-fitting since it introduces bias but reduces drastically the variance. Generally, automatic selection of features works much better than manual selection because the algorithm is able to find correlations between the features that are not always obvious even for a human expert. For example, as Guyon pointed out, “a variable that is completely useless by itself can provide a significant performance improvement when taken with others” [35].

The main feature selection algorithms are minimum redundancy maximum relevance (mRMR) and principal component analysis (PCA). The former, mRMR selects the subset of variables most relevant to the problem. The variables are ranked according to the information that they contain. This quantity of information is calculated by using the concept of entropy from information theory. The latter, PCA transforms the set of variables into a new smaller set of features. In both cases, the goal is to extract as much information as possible from as few features as possible from the dataset. While PCA has been extensively used for the problem of intrusion detection, particularly on the KDD99 dataset, surprisingly, mRMR seems not to have been used much or at all according to [23]. At any rate, feature selection is an important preprocessing step of a machine learning algorithm that should not be overlooked. In particular, it should always be applied when the problem has a high dimensionality as is the case of intrusion detection, since there is no point in

feeding an algorithm with irrelevant or information-less features.

In the intrusion detection problem, the variables used by an NIDS are information extracted from the header and payload of the packets going through the network. The list of variables used in the KDD99 dataset can be found in Appendix A. Attacks on a network can be detected by analysing these variables. However, for each type of attack, only a few variables are relevant. Since each type of attack has its own subset of useful variables, it is not optimal to apply a unique feature selection algorithm. Instead, it is better to feed different algorithms with several feature sets as will be seen in Section 3.6. In a real environment, there is a possibility to use a different sensor for each algorithm to extract the features needed. An architecture of parallel sensors has been proved to be a good design choice for powerful IDS [29]. More information about feature selection can be found in [35]. Finally, a recent review describes the main techniques applied in data preprocessing for anomaly based network intrusion [23].

3.3 Machine Learning Applied to IDS

3.3.1 Why Machine Learning?

As explained in Section 2.1.6, new attacks with unknown signatures can occur. These attacks can exploit what is called zero-day vulnerabilities [26] referring to the fact that it is the first day or day zero that the vulnerability is observed in the wild. Consequently, there is no patch available to protect the vulnerable system against an attack exploiting this vulnerability. A list of vulnerabilities that are yet to be publicly disclosed is available on the website of the Zero Day Initiative [74].

Another problem that cannot be solved by misuse-based IDSs is the variants of known attacks. As previously explained (see Section 2.1.6.1), misuse-based IDSs can only detect attacks which signatures are available in their signature database. Signatures of attacks are very specific, that is why a slight variation of the attack can be unnoticeable for misuse-based IDSs. Thus, a system should be able to recognize autonomously malicious actions in order to defend itself against zero-day vulnerabilities and variants of previously seen attacks. That is why learning mechanisms must be implemented to detect and prevent these attacks without having to wait for an update of the signature database or a patch for the vulnerable system.

As mentioned in the introduction of this chapter, machine learning enables a computer to learn how to make predictions. In the case of intrusion detection, the algorithm should be able to predict if an action on a network or a host is normal or malicious. Hopefully, this kind of system would be able to detect an attack exploiting a zero-day vulnerability or even better, prevent it, update its local database with a new signature representing the attack and distribute the signature to other signature databases on the network. Unfortunately, machine learning has weaknesses which have to be overcome in order to create an efficient, or at least, a useful intrusion detection system. The next section describes these weaknesses as well as possible guidelines to overcome them.

3.3.2 Difficulty to Apply Machine Learning to Intrusion Detection

The main challenges that machine learning algorithms have to overcome in order to be useful in the field of intrusion detection have been summarized in the excellent article written by Sommer and Paxson [70]. A set of guidelines aiming at improving the current research is also provided by the two authors. This paper is a must read for anyone aiming at developing a efficient intrusion detection system.

First of all, intrusion detection is very different from the other domains in which machine learning has been applied successfully. Indeed, spam detector or recommender systems do not require a level of precision as high as intrusion detection systems. As explained in Section 2.1.1,

both false positives and false negatives can have a very negative impact on the network that is affected by the misclassification of the IDS. FPs consume time and money because an administrator must go through all false alarms to eventually determine that nothing damaging has occurred. FNs are even more problematic because the integrity of the system is compromised although the IDS has not triggered any alarm. Thus, the cost of error is extremely high. Moreover, intrusion detection is one of the only fields where machine learning is applied and in which a user can try to evade the system or even try to control it. However, Sommer and Paxson inform that only very skilled attackers would be able to do so and since there is a very small fraction of such users on the Internet, this challenge should be addressed last.

Secondly, a more worrying problem is the fact that machine learning algorithms are designed to recognize examples similar to those available in the training set used to build the model of the data. Consequently, an IDS using machine learning would have a hard time detecting attacks which patterns are totally different from the data previously seen. In other words, even though machine learning is a suitable candidate to detect variants of known attacks, detecting zero-day vulnerabilities might be out of reach for these kinds of algorithms. Furthermore, the lack of labelled datasets is a huge problem for the research community. The only labelled datasets publicly available are the DARPA98 dataset and the KDD99 dataset which is a variant of the first one. Many criticisms have been expressed towards these two datasets as we will see in Chapter 4, Section 4.1. However, they have been intensively used by the researchers in the last decade. In fact, the confidential nature of the data which must be analysed by an IDS explains why it is so difficult for researchers to provide datasets for the community. Some of them have tried to anonymize the data, but even when using such a technique, critical information can still be extracted.

Thirdly, another crucial point noticed by Sommer and Paxson is what they call the "semantic gap". Indeed, many researchers are only interested in the application of machine learning to the problem of intrusion detection without trying to understand what information the IDS should provide to a network administrator. To content oneself with a very efficient detector somehow misses the primary purpose of the system that is being engineered. An IDS must provide relevant information about an attack that occurred on the network or that is in progress to the administrator of the system. If this information is not clear or thorough enough, the IDS will be useless and will probably not be activated at all. On top of that, the information must be relevant to the particular setting of the network that the IDS is monitoring. That is to say that the security policy of the organization that use the system must be known by the IDS to avoid unwanted false alarms. Unfortunately, most of the time security policies are too vague to be interpreted efficiently by current IDSs.

Finally, the diversity of the traffic is another factor that can mislead predictors built with some rigid machine learning models. For most of the anomaly detectors, a sudden rise of traffic volume is considered as anomalous because it diverges from the normal traffic that could have been observed in the training set. However, in most cases, extreme variations of the number of connections to a server does not necessarily mean that the server is under a `DoS` attack. For example, this abrupt change could merely be due to the final of the World Cup in football or any other event that condensate the number of connections to a single moment. In any case, it is important to understand that often simpler solutions are available to solve a particular problem and that machine learning is not the answer to every problem. Machine learning should be employed for what it is good at and should not be forced into inappropriate problems. Therefore, we are convinced, like Sommer and Paxson, that machine learning can improve tremendously current IDSs, but only if applied with care and if researchers from the security field and the machine learning field work together.

3.4 Artificial Intelligence Applied to Intrusion Detection

Intrusion detection systems have been around since the 80s. In 1980, James Anderson introduced the concept of HIDS [8]. Seven years later, Dorothy Denning laid the foundations of intrusion detection system development [25]. Then, NIDS were introduced in 1990 by Todd Heberlein [37]. As Paul Innella described, "the history of intrusion detection is as confusing as Greenspan's economic strategies" [38]. However, the turning point arises at the beginning of the 21st century with some exceptions in the late 90s, when researchers in artificial intelligence started to incorporate their algorithms to improve IDSs. Most of the research involving artificial intelligence applied to IDSs until 2007 is summarised in the excellent review paper [80]. This review starts by giving some background information about the intrusion detection field. Afterwards, the paper goes into a detailed state-of-the-art organised according to the different algorithms applied by most of the researchers.

The first algorithm being reviewed is the artificial neural network (ANN) and all its variants. An interesting variant of supervised ANN is the cerebellar model articulation controller (CMAC) which is able to learn new attacks on the fly without being retrained. This approach was adopted by Cannady [14]. Rhodes et al. and Sarasamma et al. concluded that different subsets of features should be used to detect different attacks [62, 67]. They applied their finding with an unsupervised ANN called self-organizing maps (SOM) which cluster data based on the absolute distance between them. The last unsupervised ANN type reviewed was adaptive resonance theory (ART) which seemed to outperform SOM for the problem of intrusion detection. In general, ANN have long training time and retraining problems. Moreover, the number of layers as well as the number of neurons per layer must be selected manually. In the case of unsupervised learning, the number of clusters has to be chosen as well. Wu and Banzhaf also give indications on how to improve the current ANN. In particular, "ensemble or hierarchical structure achieve better performance than single layer network" [80] and "selecting good feature sets is another way to improve performance" [80]. In this thesis, we will combine these two techniques and assess the resulting model on the KDD99 dataset.

The second algorithm being reviewed in [80] is the fuzzy sets in case of misuse detection and anomaly detection. Since intrusion types can be extremely varied, fuzzy logic increases the robustness of IDSs. As we have seen in the previous chapter, for example, `Probe` attacks are usually detected by the high number of packets sent with little or no content. But, what if the attacker fills the packets with random bytes to evade detection? In this case, flexible rules such as fuzzy sets would increase the probability of detection. An interesting application of fuzzy logic is decision fusion which combines the results of several algorithms to produce a unique fuzzy output. Ensemble approaches would benefit greatly from this property as we will see in more details in the next sections.

The next set of algorithms concerns evolutionary computation (EC). EC is a range of algorithms inspired by evolutionary behaviour that can be used successfully for the problem of intrusion detection. However, many challenges must be overcome. For example, an appropriate termination criterion should be determined to improve the accuracy without compromising the training time. The long training time is also a problem that must be faced by researchers willing to apply EC to the problem of intrusion detection. The unbalanced data distribution in the KDD99 dataset increases the difficulty of assessing the performance of the algorithms. For example, the `R2L` class is represented by only 52 examples whereas the `DoS` class is composed of almost four million examples. In fact, the skewness of the data is not only a problem for EC, but it affects all types of algorithms.

Artificial immune systems (AIS) are the next topic of the review. AIS is a type of algorithms which tries to mimic the human immune system (HIS). Researchers observed that HIS have been able to repel intrusions out of the human body for thousands of years with a very low level of

failure. This assessment lead the research community to study the properties of the HIS in order to build algorithms that could act in a similar way. Besides, it is not a surprise that researchers try to understand and apply biological mechanisms that perform so well. In the beginning of AI, researchers copied the processes occurring in the human brain to build an important class of machine learning algorithms available today called ANN. Research in AIS is young but has attracted a lot of attention in the last decade and will definitely grow in the next one. Negative selection (NS) algorithms seem to be the more widely used in the field of intrusion detection because they are based on a new type of anomaly detection in which model of non-self pattern (anomalous) instead of self pattern (normal) are built. Danger theory appears to grow rapidly and is also described in [80]. Since AIS is young, many improvements can be achieved to obtain a better detection accuracy. First of all, most of the researchers have assessed their models on benchmark datasets which do not always represent the real-world accurately. Furthermore, when modelling self patterns, the change in normal behaviour from one network to another should be taken into account. In addition, new artificial immune algorithms should be engineered to mimic more precisely the HIS. The algorithms could be enhanced to execute a immune response turning the IDS into an IPS. Eventually, [80] point out a number of reviews on AIS between 2006 and 2007.

Another very powerful artificial intelligence technique is swarm intelligence (SI). Wu and Banzhaf describe the recent advances in the field focusing on the main types of SI algorithms: ant colony optimization (ACO) and particle swarm optimization (PSO). SI techniques are bio-inspired algorithms taking advantage of optimization mechanisms observed in nature. For example, ACO was developed by mimicking social behaviours of colonies of ants and termites whereas PSO takes its inspiration from the motion of flocks of birds and schools of fish. SI techniques are well-known for their speed and accuracy. They exploit a number of agents which work in parallel in order to search for a solution. Consequently, SI is very suitable for intrusion detection because of the huge datasets and high dimensionality. Swarm intelligence in intrusion detection is also reviewed in more details in [41] which extends the work of [80] to 2010. They compare the performance of the different SI algorithms on the KDD99 test set.

Last but not least, soft computing is also reviewed by [80]. Generally speaking, soft computing is a relatively new paradigm that is used to reduce the imprecision of individual algorithms. Two main types of soft computing techniques are currently available depending on the degree of coupling between the algorithms. Systems engineered with high coupling between algorithms form the class of hybrid systems in which each algorithm cooperates with the others to increase the accuracy of the IDS and cannot be replaced at the risk of losing the coherence of the system. Loosely coupled systems assemble several algorithms together in a way that does not constrain the particular type of each algorithm. Consequently, each algorithm can be replaced easily by another more suitable one. Ensemble approaches is currently the only representative of this class of systems. However, since ensemble approaches is a fairly new technique applied to intrusion detection, their description in the review is somewhat limited. For this reason, a more detailed state-of-the-art concerning ensemble approaches is done in Section 3.6.2.

The review also provides a comparative table of the performances of the different families of algorithms. These evaluations are done on the KDD99 test set. This is important because some researchers use a subset of the KDD99 training set as the test set leading to higher accuracy. Indeed, the KDD99 test set is composed of a high number of new attacks unseen in the corresponding training set. Assessing the performance on the KDD99 test set truly shows the generalization power of the algorithm and in general, all algorithms perform poorly on the `U2R` and `R2L` classes. The best results achieved until 2007 are obtained by genetic programming (GP) using Transformation functions for `R2L` and `Probe` with 80.22% and 97.29% accuracy respectively and by linear genetic programming (LGP) for `DoS` and `U2R` with 99.7% and 76.3% accuracy respectively.

Another interesting review focuses mainly on machine learning techniques applied to intrusion

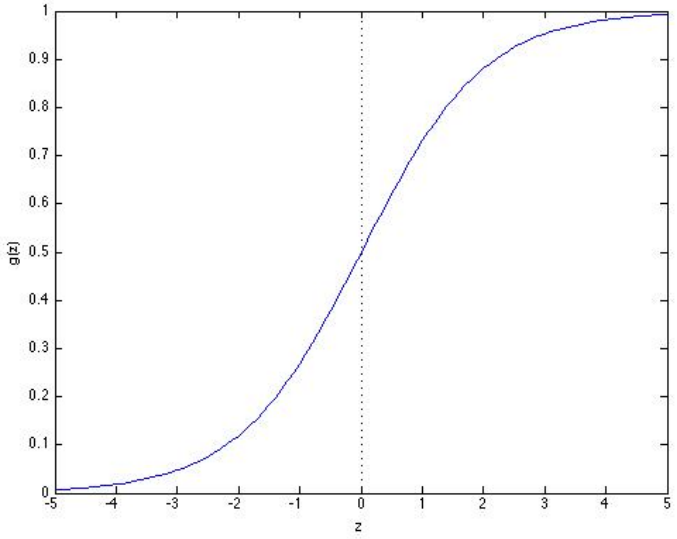


Figure 3.1: Logistic function

detection between 2000 and 2007 [76]. In particular, Tsai et al. compare the number of papers published concerning single classifiers, hybrid systems and ensemble approaches. They also show which dataset was used and if feature selection was applied for each of the papers. They conclude that comparisons of hybrid systems and ensemble classifiers is necessary, that combinations of ensemble and hybrid approaches should be studied and that it is not currently known which method of feature selection performs best.

Despite the important improvements that were achieved by the AI research community in the field of intrusion detection, almost no commercial IDS uses machine learning techniques. The main reason is the number of false positives and false negatives that is too high for a real-world application. Companies on the IDS market include Cisco, IBM and TippingPoint. Their websites are a mine of information on the topic which is in general up-to-date. This is a good starting point to find out what the real-world IDSs need to improve because these companies experience the IDS problem in their products.

3.5 Machine Learning Algorithms

3.5.1 Support Vector Machines (SVM)

Support vector machines is one of the most widely used machine learning algorithms [55]. SVM can be seen as a more elaborated version of logistic regression. Consequently, to understand SVM, one should start by understanding logistic regression. Although it is possible to use SVM as a multi-class classifier, usually this algorithm outputs binary values. In this case, an hypothesis must be formulated in order to output either 0 or 1. A natural candidate is the logistic function (see Figure 3.1).

$$g(z) = \frac{1}{1 + \exp(-z)}$$

In machine learning, the notation of the logistic function is slightly modified.

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

where

- x is a matrix of dimension $m \times n$ where m is the number of examples in the dataset and n is the number of variables in the dataset.
- θ is a vector of parameters of dimension $n \times 1$ where n is the number of variables in the dataset. The goal of the training phase of the machine learning algorithm is to determine the optimal values for this vector of parameters.
- $h_\theta(x)$ is called the hypothesis and represent the value that the classifier predicts for a particular example $x^{(i)}$.

When using supervised learning (recall that the labels of the training set is the vector y of dimension $m \times 1$ where m is the number of examples in the training set), we want that

$$h_\theta(x^{(i)}) = \begin{cases} 1 & \text{if } y^{(i)} = 1 \\ 0 & \text{if } y^{(i)} = 0 \end{cases}$$

Figure 3.1 shows that $h_\theta(x^{(i)}) = 1$ if $\theta^T x^{(i)} >> 0$ and $h_\theta(x^{(i)}) = 0$ if $\theta^T x^{(i)} << 0$.
These conclusions lead to the following cost function for the logistic regression

$$\sum_{i=1}^{m} (-y^{(i)} log(h_\theta(x^{(i)})) - (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))) + \frac{\lambda}{2} \sum_{j=1}^{n} \theta_j^2$$

or

$$\sum_{i=1}^{m} (\underbrace{-y^{(i)} log(\frac{1}{1 + \exp(-\theta^T x^{(i)})})}_{Cost_1(x)} \underbrace{-(1 - y^{(i)}) log(1 - \frac{1}{1 + \exp(-\theta^T x^{(i)})})}_{Cost_0(x)}) + \frac{\lambda}{2} \sum_{j=1}^{n} \theta_j^2$$

The last term of the equation is a regularization term that helps reducing overfitting. The objective of the machine learning algorithm is to minimize this cost function to find the vector of parameters θ that represents the data as well as possible. As expected, when $y^{(i)} = 1$ and $h_\theta(x^{(i)}) = 1$ or when $y^{(i)} = 0$ and $h_\theta(x^{(i)}) = 0$, the cost function is equal to 0 whereas when $y^{(i)}$ and $h_\theta(x^{(i)})$ have opposite values, the cost has a high value since $log(0) = -\inf$. These results can be observed more clearly on Figures 3.3 and 3.2 representing the cost function when $y^{(i)} = 0$ and $y^{(i)} = 1$, respectively.

These figures also show the difference between logistic regression and SVM. The cost function for logistic regression is drawn in blue and the one for SVM is drawn in red. First, the cost function for SVM approximates the cost function of logistic regression with several linear segments. This makes SVM computationally more efficient than logistic regression. Second, the cost function for SVM is equal to zero only if $\theta^T x \geq 1$ and y = 1 or if $\theta^T x \leq -1$ and y = 0. These conditions are stricter than the ones for logistic regression where we wanted $\theta^T x \geq 0$ when y = 1 and $\theta^T x < 0$ when y = 0. That is the reason why SVM is sometimes called a large margin classifier. Finally, the SVM decision boundary can be obtained by solving the following minimization problem:

$$\min_\theta (0 + \frac{\lambda}{2} \sum_{j=1}^{n} \theta_j^2)$$

under the constraints that

- $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$
- $\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

So far, we have seen how to find a decision boundary using the SVM algorithm when the data is linearly separable. However, if this is not the case, the notion of kernels must be introduced. The kernel function measures the similarity between the examples in the dataset. There are different types of kernels: Gaussian kernel, polynomial kernel, etc. As a result, the non-linear boundary will group examples which are similar to each other.

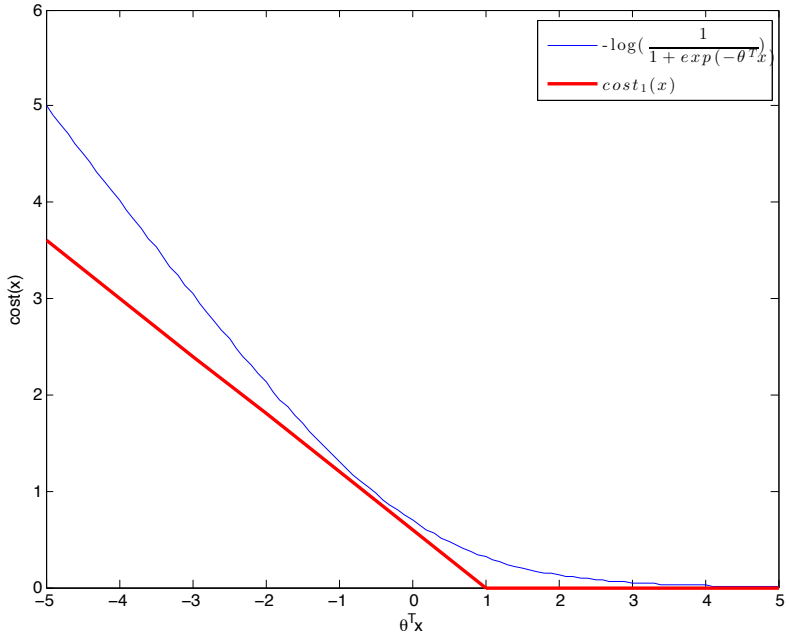


Figure 3.2: Cost function when $y = 1$

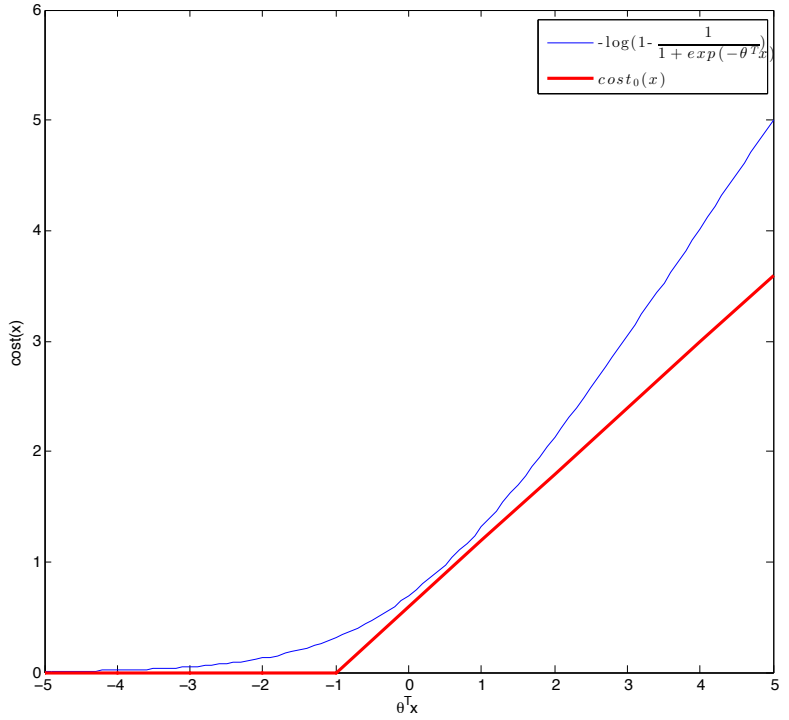


Figure 3.3: Cost function when $y = 0$

3.5.2 Linear Genetic Programming (LGP)

Genetic programming (GP) is a subclass of evolutionary algorithms [11, 42]. Similarly to other artificial intelligence paradigms such as swarm intelligence, evolutionary computation is a bio-inspired, population-based technique. However, as the name suggests, EC is inspired by the

theory of evolution. The individuals in the populations are often called chromosomes and the pieces of these chromosomes that are modified during the evolutionary process are called genes. Every evolutionary algorithm follows the same basic scheme.

The first step is to initialize a population of solutions with random values. Then, similarly to the evolutionary process, natural selection occurs. This selection varies from one implementation to another, but generally a portion of the population representing the set of best current solutions is selected. A solution is considered better than another one if its fitness function has a higher value. The fitness function depends greatly on the problem being studied. The selected solutions are then evolved by mimicking natural reproduction, chromosomal crossovers and genetic mutations. The new individuals resulting from this evolution are added to the previously selected group to form the new population. The fitness function of each individual of the new population is computed. If the termination criterion is not met, the evolutionary process proceeds to another iteration. Otherwise, the best individual of the current population represents the solution of the problem which may not be optimal depending on how well the fitness function and the termination criterion are defined. The diversity of the individuals and the recombinations provide an efficient way to cover the search space. The selection of the fittest is supposed to allow the algorithm to converge to an optimal solution.

The mechanisms used in GP are similar to those of evolutionary algorithms. However, the purpose of GP is well-focused. In genetic programming, the individual solutions are executable programs. A program is defined by a sequence of instructions. At each iteration of the algorithm, the sequence of instructions of the best individuals are modified in the evolutionary fashion described above. In other words, GP is a machine learning technique which enables computers to learn how to program. Obviously, GP is not restricted to this task and can be used in other applications such as intrusion detection since the core of the method is very general.

Finally, linear genetic programming is a variant of GP in which the instructions are part of an imperative language such as `C++` or `Java` [11]. Earlier versions of GP were based on instructions belonging to functional programming languages such as `Erlang` or `Haskell`. Programs developed with functional programming languages can be represented as a tree in which the nodes are the functions and the leaves are the constants and the input values. It is important to understand that the term linear in the name LGP is connected to the structure of the programs written in imperative programming language. It has nothing to do with the model built by LGP which is generally non-linear.

3.5.3 Multivariate Adaptive Regression Splines (MARS)

Multivariate adaptive regression splines is an adaptation of linear regression to non-linear models. It is a flexible multivariate non-parametric regression modelling technique. The problem with linear regression or even polynomial regression is that the developed model will only be accurate if the underlying function, represented by the data, is linear or polynomial respectively — which is not always the case in practice. However, linear and polynomial models have also advantages over more complex methods. They are fast to compute, easy to interpret and do not require a huge dataset. The following description of MARS gives only a general overview of the technique allowing the reader to understand the basic concepts used in this algorithm. For a deep analysis of multivariate adaptive regression splines, refer to [30].

The technique used to overcome the drawbacks of linear and polynomial regressions is piecewise parametric fitting which determines several simple parametric functions defined over a different subregion of the domain of the data. To determine the subregions, recursive partitioning regression is used. The procedure starts with one region which includes the entire domain. At each iteration, the subregions are split into two subregions. A step function is defined for each subregion and the criterion used to assess the new model at each iteration is the goodness-of-fit. The

problem of recursive partitioning is that it introduces discontinuities at the intersections between the subregions, also called knots. This problem can be solved by replacing the step function with a continuous function called spline.

Splines are polynomial functions that are piecewise-defined and are smooth at the knots. The algorithm MARS includes the previously mentioned improvements to build more accurate models. The subregions are tensor products of $K + 1$ intervals where K represents the number of knots. The strategy is to overfit the data with a model including a large number of knots, and then to fuse the subregions back together in a backwards stepwise fashion until a termination criterion is met. Thanks to the use of splines, the subregions are overlapping each other, hence avoiding zero values predictions at the knots. In the MARS strategy, the parent function is not removed after it has been split into two subregions. This makes the parent and the children available for more splitting. The final model built by MARS includes a set of splines that fit non-linear data much better than a straight line or a single polynomial function.

3.5.4 Decision Tree (DT)

Decision tree is one of the simplest and most intuitive machine learning algorithm used for classification. Despite its simplicity, it is a powerful tool for decision making. As the name suggests, DT builds a tree-like structure based on a set of labelled data. Each node of the tree represents one variable of the dataset and the leaves represent the different labels of the examples. Each branch growing down a node of the tree represents a possible value for the variable contained in the node. If the variables have continuous values, the branches represent non overlapping intervals of values contained in the variable space. Consequently, to make a prediction on an unseen example, one must start from the root node of the tree and follow the branches corresponding to the values of the variables of the example until reaching a leaf. The ending leaf gives the class of the example for which the prediction was needed.

There are many ways to determine the order of the variables in the tree. For example, a popular technique from the information theory is to calculate the information gain of each variable and then to rank the variables according to these values. The variable with the highest information gain is the root, the one with the second highest information gain is the second node, etc. For a binary tree, the information gain can be calculated with the following equation

$$gain(A) = I(p, n) - E(A)$$

where

- E(A) is the expected information required for the tree with A as root

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

  where

  - `p` is the number of positive examples
  - `n` is the number of negative examples
  - p_i is the number of positive examples for the `ith` branch
  - n_i is the number of negative examples for the `ith` branch
  - `v` is the number of branches from the root A

- I(p,n) is the information required for classification

$$I(p, n) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

  where

- `p` is the number of positive examples
- `n` is the number of negative examples

Unfortunately, this technique has been proven to give higher score to variables with many values which does not necessarily mean that those variables are important. For example, a variable which has a different value for each examples of the dataset does not give any information to classify the example, but will have a high value for the information gain. For this reason, several variants of this selection criterion have been developed to improve the construction of decision trees. They are all based on information theory, but their description is out of the scope of this thesis. The same mechanisms of the information theory can also be used to select features as we have seen in Section 3.2. Many algorithms have been developed to build decision trees. ID3 is an example of a family of algorithms based on decision trees, described in details in [60]. The decision trees with the highest accuracy are often the smallest ones because they have better generalization power than more complex trees. DT is a fast and efficient algorithm that has been applied successfully to numerous classification tasks.

3.6 Ensemble Approaches

The ensemble approach is a relatively new trend in artificial intelligence in which several machine learning algorithms are combined. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Ensembles are particularly useful when a problem can be divided into subproblems. In this case, each module of the ensemble, which can include one or more algorithms, is assigned to one particular subproblem. As we have seen in the previous chapter, network attacks can be divided into four classes: denial of service, user to root, remote to local and probe. One module of the ensemble designed in this work is itself an ensemble of decision trees and is specialized on the detection of one class of attacks. In the same way that web-application firewalls (WAF) address the problem of web-application attacks, we want IDSs to have specific tools to defend against the major classes of attacks. Because signatures of `DoS` attacks are very different from the ones of `U2R` attacks, it is natural to have a different set of features and a different algorithm to detect the two classes of attacks.

> "A single IDS can cover only a a limited number of different types of input data and can identify only a limited number of attacks."
>
> Christopher Kruegel et al. [43]

Ensembles are also a way to build different types of approaches to solving the same problem. This use of ensembles is analogue to the process in which a person requires a medical diagnosis. Usually, if the person is diagnosed with some serious illness, such as a cancer, he or she will try one or more other doctors to have a different opinion on the matter. This cross-validation step increases the probability to obtain an accurate diagnosis. If the majority of the experts confirm the first diagnosis, then it is more likely that the person has a cancer. Similarly, the outputs of several algorithms used as predictor for the intrusion detection problem are combined to improve the accuracy of the overall system. The inner structure of each module of the ensemble developed in this work is composed of several decision trees which are trained with different sets of features to increase the accuracy of the IDS. In this thesis, the efficiency of using different sets of features for each class of attacks is assessed. To facilitate the evaluation, all algorithms composing the ensemble are decision trees. However, in future work, it would be interesting to try different algorithms to improve the accuracy even further.

The difficulty of ensemble approaches lays in the choice of the algorithms constituting the ensemble and the decision function which combines the results of the different algorithms. Of course, the more algorithms the better, but it is important to take into account the computational

expense that is added by each new algorithm. The decision function is often a majority vote which is both simple and efficient. Nevertheless, it could be interesting to analyse alternatives to obtain an optimal combination. Another advantage of ensemble approaches is their modular structure, unlike hybrid constructions which are engineered with algorithms that usually have non-interchangeable positions. Consequently, the designer of an ensemble can easily replace one or more algorithms with a more accurate one.

3.6.1 Bagging vs. Boosting

Bagging and boosting are the two main techniques used to combine the algorithms in an ensemble. In an ensemble using the boosting technique, the algorithms are used sequentially. The first algorithm analyses all the examples in the dataset and assigns weights to each of them. The examples with a higher value for the weight are the ones that were classified wrongly by the algorithm. Then, the next algorithm receives as an input the dataset as well as the weights for all examples in the dataset. The weights allow the algorithm to focus on the examples that were the most difficult to classify. These weights are updated according to the results of the second algorithm and the process moves to the third algorithm. This sequence continues until the last algorithm of the ensemble has processed the data. The advantage of this technique is that the most difficult examples can be classified correctly without adding too much computational overload. The use of weights, which are updated throughout the process, reduces the computation time as the data goes down the chain of algorithms.

In an ensemble using the bagging technique all algorithm of the ensemble are used in parallel. In this case, each algorithm builds a different model of the data and the outputs of every predictors are combined to obtain the final output of the ensemble. In order to build different models, either each algorithm of the ensemble, or the data fed to each algorithm, or both, can be different. In this thesis, only the data fed to each algorithms are different, but all algorithms are decision trees as explained above. Since all algorithms perform in parallel, each of them can be executed on a different processor to speed up the computation. This is an important advantage on the boosting technique because nowadays multicore processors are very common even in personal computers. With this kind of architecture, the ensemble does not increase significantly the time of computation compared to a single algorithm because the only additional time needed is used for the decision function which combines the outputs of all algorithms.

3.6.2 Ensemble Approaches Applied to Intrusion Detection

Ensemble approaches were introduced for the first time in the late 80s. In 1990, Hansen and Salamon [36] showed that the combination of several ANNs can improve drastically the accuracy of the predictions. The same year, Schapire [68] showed theoretically that if weak learners are combined, it is possible to obtain an arbitrary high accuracy. Weak learners are classifiers able to classify correctly only a small fraction of the examples in a dataset. Following these two main works on the topic, the research community actively studied ensembles during the 90s. However, this paradigm was used for the first time for intrusion detection in 2003. The majority of the papers written on the topics were found between 2004 and 2005. Recently, there has been a regain of interest for ensembles in the field of intrusion detection [9, 27, 28, 31, 82].

Mukkamala et al. [52] showed that an ensemble composed of different types of ANN, SVM with radial basis function (RBF) kernel and MARS combined with the bagging techniques outperforms more traditional approaches with a single algorithm. The variants of ANN were: resilient back propagation (RP), scaled conjugate gradient algorithm (SCG) and one-step-secant algorithm (OSS). The experiments were performed on a subset of the DARPA1998 dataset composed of 11,982 examples randomly selected from the original dataset with a number of data for each class proportional to the size of the class except for the smallest class which was included entirely.

This dataset was then divided into a training set of 5,092 examples and a test set of 6,890 examples. Five SVM and Five MARS were used as binary classifiers to classify examples between each class of attacks. The ensemble was compared to the results obtained by each algorithm executed separately in order to prove their assertion. The results show that SVM used alone outperforms the other single algorithm but is totally outperformed by the ensemble of ANN, SVM and MARS. This ensemble surprisingly obtained a 100% accuracy on the test set for the `R2L` class. However, the researchers warn that some of these results might not be statistically significant because of the unbalanced dataset used.

Chebrolu et al. explored the combination of bayesian networks (BN) and classification and regression trees (CART) in a ensemble using bagging techniques as well as the performance of the two algorithms when executed alone [18, 19]. The dataset used in this work was the DARPA 1998 from which a subset was selected in the same way as in the previous work described above. Feature selection was also applied to speed up the computation. First, BN and CART were evaluate separately with different sizes for the set of features. The performance on the set of 41 features was compared to a set of 12 selected by BN, 17 selected by CART and 19 features selected by another study. BN performed worse with a smaller set of features except on the `Normal` class. However, when using the set of 19 features, BN and CART complemented each other to increase the IDS accuracy for all classes. The final ensemble was composed of one CART in charge of detecting `Normal` examples and trained with the set of 12 features, one CART in charge of detecting `Probe` examples and trained with the set of 17 features, one CART in charge of detecting `U2R` examples and trained with the set of 19 features, one ensemble of one CART and one BN in charge of detecting `R2L` examples and trained with the set of 12 features and finally, one ensemble of one CART and one BN in charge of detecting `DoS` examples and trained with the set of 17 features. The approach used by Chebrolu et al. is very similar to the one used in this thesis. However, the lack of error analysis could be deplored.

Abraham and Jain investigated different machine learning algorithms for the problem of intrusion detection [4, 5]. Fuzzy rule-based classifier, DT, SVM, LGP and an ensemble were evaluated on the DARPA 1998 dataset. A subset of this dataset was selected in the same way as in the two previous works mentioned above. Feature selection was also applied to reduce the number of variables of the dataset to 12. Fuzzy rule-based classifier outperformed all other methods when trained with all 41 features with the second set of rules FR_2 scoring 100% accuracy for all classes of attacks. LGP seemed more appropriate when using a smaller set of features except for the `U2R` and `Normal` classes. Finally, the ensemble was composed of one DT in charge of the `Normal` instances, one LGP for `Probe` and one LGP for `DoS`, one fuzzy set of rules FR_2 for `U2R` and one LGP for `R2L`. The results obtained with the ensemble are very encouraging with more than 99% accuracy for all classes.

Abraham and Johnson [6] extended the work previously carried out in [18]. A hybrid model composed of SVM and DT was added to the previous ensemble. This new model works as follows. The data is first sent to the DT which generates a tree to fit the features and values of each examples in the dataset. The tree generated is then sent to the SVM to produce a final output. The information contained in the node should improve the detection of the SVM. A single DT is in charge of detecting `U2R` attacks, a single SVM is in charge of detecting `DoS` attacks, the hybrid model is in charge of `Normal` instances and the ensemble built in [18] is in charge of `Probe` and `R2L` attacks. Given the results, the hybrid model does not seem to help very much. [58] describe the same work as [6]. However, results of DT alone are also given in that paper.

Abraham et al. explored genetic programming and evaluated the performance of linear genetic programming (LGP), multi-expression programming (MEP) and gene expression programming (GEP) on the DARPA 1998 dataset [2, 3]. The dataset was again sampled in the same way as in the previous works mentioned in this section. Their results showed that LGP obtained the highest accuracy for the `Probe` and `DoS` classes whereas MEP obtained the highest accuracy for the other

classes. Even though the paper does not mention any use of ensemble, the work clearly shows the high potential of genetic programming algorithms and therefore, they should be considered as part of an ensemble.

Zainal et al. compared the results of several machine learning algorithms [81] . In particular, the performance of linear genetic programming (LGP), adaptive neural fuzzy inference system (ANFIS) and random forest (RF) were analysed. The dataset used in this work was the KDD99 dataset from which a subset was extracted in the same way as in the other experiments. Different sets of features were selected for each class of attack from the 41 features available in the datasets. Furthermore, an ensemble was also engineered by combining the ANFIS, LGP and RF algorithms. However, the exact configuration of this ensemble is not described in the paper. The bagging technique was probably used to form the ensemble. Results of individual algorithms as well as the results obtained by the ensemble were given. The ensemble outperformed the single algorithms.

Folino et al. examined the performance of a distributed system of ensembles called GEdIDS [27, 28]. The system is composed of several genetic programming (GP) ensembles distributed on the network based on the island model. Each ensemble is trained with a different dataset in a number of T rounds. Once the ensemble is trained in one round, it is exchanged with the other islands through the distributed environment called dCAGE (distributed Cellular Genetic Programming System). The weights of the tuples are then updated and the next round takes place. The dataset used was the KDD Cup 1999. The advantage of a distributed system pointed out by the researchers is the increase in privacy and security in comparison to a central IDS which has to collect audit data from different nodes on the system. The technique used to combine the individual GP algorithms was AdaBoost which is a kind of boosting method for ensembles. The use of an ensemble approach allowed the researchers to deal with a large dataset, to modify the system easily and to obtain a robust classifier. The results obtained by the system are average for the `Normal` class, the `Probe` class and the `DoS` class and very low for the `U2R` and `R2L` classes. However, very few papers study distributed environment for intrusion detection even though this might be a very good idea for the reasons mentioned above. Finally, a comparison with other boosting methods is also provided. Folino wrote many other papers related to GP ensemble but unrelated to the problem of intrusion detection.

Peng Zhang et al. evaluated the robustness of ensembles when confronted with noisy data streams [82]. Noisy data streams are defined by the researchers as having examples with incorrect labels. The solution to noisy data proposed in this work was an aggregate ensemble (AE) which is able to tolerate imprecision and errors. AE builds several classifiers on different sets of data using different learning algorithms. AE was then compared to Horizontal Ensemble framework in which classifiers are built on different sets of data but with only one learning algorithm by set of data and Vertical Ensemble framework in which several classifiers are built on the new set of data with different learning algorithms which are then combined into an ensemble. Both synthetic and real-world data streams were used to evaluate the models. The algorithms used in the AE were SVM, DT and logistic regression. It seems that AE outperforms both the horizontal and vertical Ensemble frameworks.

Bahri et al. introduced a new method based on Greedy-Boost [9]. The ensemble was again based on the boosting technique and was an adaptation of Adaboost that makes it resistant to noise. For one thing, the greedy-boost classifier is a linear combination of models instead of a simple model as in AdaBoost. For another, the distribution of weights is updated according to the initial distribution instead of the previous one. Greedy-Boost was trained on several dataset. First, the `KDD99` dataset was used to prove that the model could reduce significantly the false negative rate (FNR). Then, it seems that Greedy-Boost was tested on three different datasets: Attack-Free Farpa dataset, Attack-Free Gatech dataset and HTTP-Attack dataset. But, the results displayed include only the ones for the `KDD99` datasets. These results are extremely high in terms of precision and recall. In particular, the precision of the most difficult class (`R2L`) is much higher than what

is usually observed. However, the researchers do not state if they used the test set, the training set or a modified version of one of the sets to evaluate the model.

Gonzalez et al. examined the performance of nine different ensembles on real-life datasets when applying a mutant operator to modify slightly the packets of a `Probe` attack [31]. In particular, datasets 6 and 9 are the ones on which the experiments were performed. In dataset 6, both the time and the number of scans were modified. All examples in dataset 6 were classified correctly by all ensembles. For dataset 9, both the amount of packets and the destination ports were modified. MultiboostAB with REPTree as a classifier, RandomSubSpace with REPTree as a classifier, RandomSubSpace with SImpleCart as a classifier, AttributeSelectedClassifier with SImpleCart as a classifier and Bagging with REPTree as a classifier obtain 100% accuracy.

This section has given an overview of the research focusing on ensemble approaches over the past decade. The list of papers is not exhaustive, but is an effort to summarise the main developments that ensemble approaches have followed in the field of intrusion detection. In particular, papers that were covered in the reviews mentioned in Section 3.4 were not included again in this section. The key conclusion that can be extracted from all these works is that the ensemble approach generally outperforms traditional approaches in which only one algorithm is used. An ensemble is a very efficient way to compensate for the low accuracy of a set of weak learners. Moreover, feature selection should provide specific subsets to train algorithms specialised in the detection of one particular class of attacks. Section 4.2 from the next chapter describes the model that was designed in this thesis taking into account all recommendations from the different papers reviewed in both state-of-the-art of machine learning and ensemble approaches applied to the problem of intrusion detection.

Chapter 4

Experiments

4.1 The KDD99 Dataset

The KDD cup 1999 dataset set, also abbreviated as the KDD99 dataset, was used for the first time in the third international knowledge discovery and data mining tools competition in 1999. This dataset is based on the DARPA98 dataset which was build by the Defence Advanced Research Projects Agency (DARPA) in 1998 during the DARPA98 IDS evaluation program. The DARPA98 dataset includes 7 weeks of data captured in the form of tcpdump from traffic passing through a network engineered for the purpose of the DARPA program. In other words, the traffic was generated in a simulated and controlled environment.

The KDD cup 1999 training set contains 4,898,431 entries. Each entry is represented by 41 variables such as duration, src_bytes, dst_bytes, etc., and a label. From these 41 variables, 3 are non-numerical: "protocol_type", "service" and "flag". There are 3 protocol types (TCP, UDP and ICMP), 70 services and 11 flags. These non-numerical variables are transformed into numerical ones to ensure that all the machine learning algorithms will be able to process their values. For example, each service name represented in the service variable is replaced by a number from 1 to 70. All the examples are separated into four different classes of attacks and the class `Normal`. The distribution of examples over the different classes is shown in Figure 4.1 and Table 4.1 (on page 40 and 41, respectively). The training set is highly unbalanced. In particular, the classes `U2R` and `R2L` are the least well represented with only 52 and 1,126 examples respectively, whereas the `DoS` class contains 3,883,370 examples. With such a small number of examples as in the `U2R` and `R2L` classes, it can be expected that it will be difficult for the classifier to predict the correct classes of unseen examples.

The analysis of the test set can also reveal interesting facts. The test set is composed of 311,029 entries. The distribution of examples containing previously unseen attacks and examples containing previously seen attacks for each class of attacks is shown in Table 4.2. The names of the unseen attack types are listed in Table 4.3. These tables show clearly that the number of unseen attacks added in the test set is huge, especially for the classes `U2R`, `R2L` and `Probe` with 44.29%, 63.34% and 42.94% respectively. Finally, Figure 4.2 shows the distribution of the examples in the test set over the different classes. This distribution is very similar to the distribution of the training set. However, the number of examples belonging to the class `R2L` is more than ten times higher than in the training set. This means that in order to perform well on the test set, the predictor must acquire a very high power of generalisation with 1,126 training examples. Another noticeable fact is that the attacks "spy" and "warezclient" belonging to the class `R2L` are not represented in the test set. In particular, "warezclient" attacks count for more than 90% of the `R2L` training set. Finally, two errors have been observed in the test set. Entries 136,489 and 136,497 have a value for the "service" variable equal to "icmp" which is erroneous. For this reason, they were removed

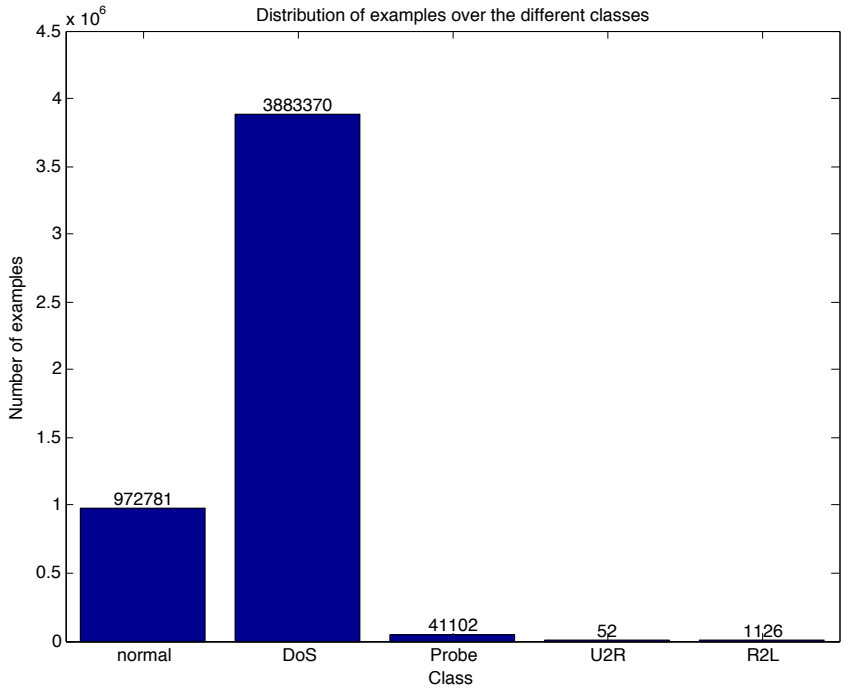


Figure 4.1: KDD cup 99 training set: Distribution of examples over the different classes (scale: $\times 10^6$)

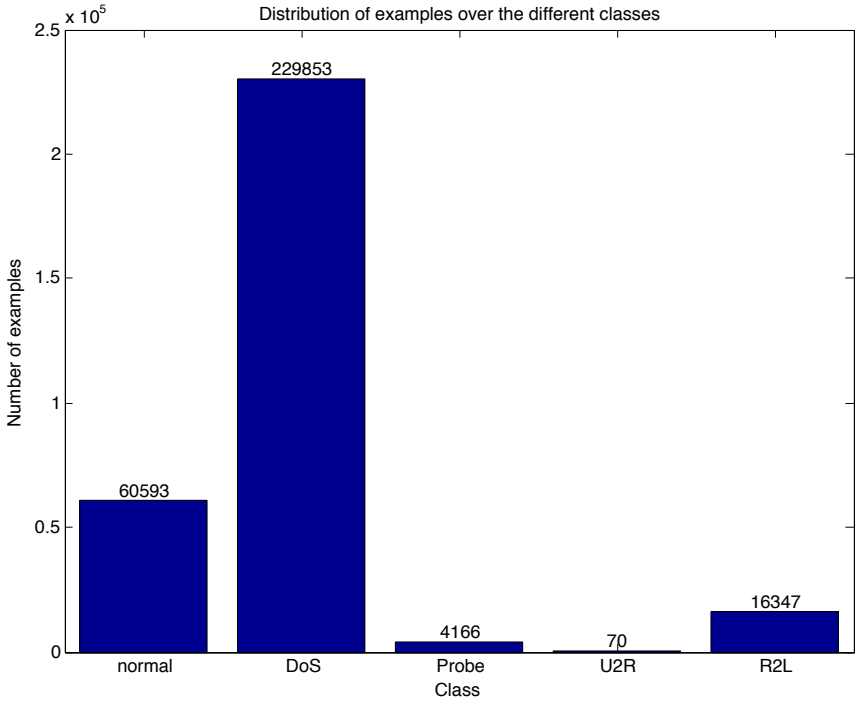


Figure 4.2: KDD cup 99 test set: Distribution of examples over the different classes (scale: $\times 10^5$)

from the test set before the experiments. However, it seems that these errors were previously

Probe		U2R		R2L		DoS		Normal	
Attack	Size	Attack	Size	Attack	Size	Attack	Size	Attack	Size
satan	15,892	buffer_overflow	30	ftp_write	8	back	2,203		
portsweep	10,413	loadmodule	9	guess_passwd	53	land	21		
nmap	2,316	perl	3	imap	12	neptune	1,072,017		
ipsweep	12,481	rootkit	10	multihop	7	pod	264		
				phf	4	smurf	2,807,886		
				spy	2	teardrop	979		
				warezclient	1020				
				warezmaster	20				
Total	41,102	**Total**	52	**Total**	1,126	**Total**	3,883,370	**Total**	972,781

Table 4.1: KDD cup 99 training set: Types of attacks in the training set over the different classes. Some attacks are marked in red when they count for almost all examples of the corresponding class.

reported [73].

Class	Nr of unseen attacks	Nr of seen attacks	Total
Probe	1,789 (42.94%)	2,377	4,166
U2R	31 (44.29%)	39	70
R2L	10,354 (63.34%)	5,993	16,347
DoS	6,555 (2.85%)	223,298	229,853
Normal	0	60,593	60,593
Total	18,729 (6.02%)	292,300	311,029

Table 4.2: KDD cup 99 test set: Distribution of unseen/seen attacks over the different classes

Class	Unseen attacks types	Total
Probe	mscan, saint	2
U2R	ps, xterm, sqlattack	3
R2L	xlock, snmpgetattack, httptunnel, named, sendmail, snmpguess, worm, xsnoop	8
DoS	udpstorm, apache2, mailbomb, processtable	4
Total		17

Table 4.3: KDD cup 99 test set: Unseen attack types

The major criticisms of the KDD99 dataset include the unbalanced distribution of the data, the redundant records which can introduce a bias in the learning phase because of their frequency, the fact that the dataset includes old attacks which have been mostly mitigated and the fact that the data were captured from a controlled environment somewhat different from what is observed in the wild. The first and second problems can be partly solved by sampling appropriate sets of examples in each class of attacks. However, the distribution of `R2L` attacks in the training set and the test set is a problem which is difficult to overcome. A solution which was not applied in this work could be to join both the `R2L` training set and `R2L` test set, shuffle the resulting dataset and divide this new dataset into a new training set and a new test set. This solution is not optimal, but could be a quick fix for the time being. Obviously, an old dataset cannot cover recent examples of attacks observed in the wild. This is a problem because new types of attacks are frequently developed by hackers. Nevertheless, the KDD99 dataset is not as useless as it seems. First of all, it can be argued that if an IDS using machine learning does not perform well on old

Probe		U2R		R2L		DoS		Normal	
Attack	Size	Attack	Size	Attack	Size	Attack	Size	Attack	Size
satan	1,633	buffer_overflow	22	ftp_write	3	back	1,098		
portsweep	354	loadmodule	2	guess_passwd	4,367	land	9		
nmap	84	perl	2	imap	1	neptune	58,001		
ipsweep	306	rootkit	13	multihop	18	pod	87		
mscan	1,053	ps	16	phf	2	smurf	164,091		
saint	736	sqlattack	2	httptunnel	158	teardrop	12		
		xterm	13	warezmaster	1,602	apache2	794		
				named	17	mailbomb	5000		
				xsnoop	4	processtable	759		
				sendmail	17	udpstorm	2		
				snmpgetattack	7,741				
				snmpguess	2,406				
				worm	2				
				xlock	9				
Total	4,166	**Total**	70	**Total**	16,347	**Total**	229,853	**Total**	60,593

Table 4.4: KDD cup 99 test set: Types of attacks in the test set over the different classes. Some attacks are marked in red when they count for almost all examples of the corresponding class.

attacks provided that the data are well sampled, why would it on newer ones. Consequently, to be useful, an IDS should at least perform well on known attacks. Otherwise, it is not even worth trying to apply it to new variants and more complex attacks. Furthermore, most of the research in the field of machine learning applied to intrusion detection uses the KDD99 dataset, making this dataset a vector of comparison between the different approaches developed by researchers. The final criticism related to the controlled nature of the environment in which the data were captured is probably the most difficult to discuss. For example, the high number of attacks in comparison to normal traffic observed in the dataset does not reflect the reality of a network in which 99.99...% of the traffic is normal. Again, appropriate sampling is required. Also, the IDS should be at least accurate on data produced by a simulated environment before being tested on a real network where the traffic pattern is probably less predictable.

4.2 Final Model

Figure 4.3 (on page 44) shows the model for the ensemble used in this thesis. The network packet being analysed is sent to four different modules: `Probe` module, `R2L` module, `U2R` module and `DoS` module. The packet goes through a data preprocessing unit (DPU). The DPU is in charge of extracting a number of features from the packet. The set of features varies depending on the module. The figure displays the number of features needed in the sets selected by [50].

The features extracted are then dispatched to different decision trees which have been previously trained with these same features on a training set. Each decision tree is a binary classifier which outputs 0 if the packet is considered normal traffic and 1 if the packet is classified as anomalous. A vector of dimension $n \times 1$ containing the output of n classifiers is then fed to the module decision function. In the figure, n is equal to 4, but it could be any number of algorithms.

The decision function of the module combines the results of each algorithm of the corresponding module and outputs a value describing if the packet is considered by the ensemble to be normal traffic or anomalous traffic belonging to the class of attacks represented in the module. Finally, a vector of dimension 4×1 containing the output of each module is fed to the ensemble decision function. This decision function combines the results and outputs a value describing if the packet is considered normal or anomalous, and if anomalous from which class of attacks.

The easiest situations are obtained when the outputs of all modules are equal to `Normal` or

the outputs of all modules are `Normal` except one. In the former case, the systems classifies the packet as normal. In the latter, the systems classifies the packet as anomalous and is able to identify unambiguously the class of attack concerned. If more than one module classify the packet as anomalous, it will be more difficult for the network administrator to understand which class of attack the anomalous packet belongs to.

The resulting model is an ensemble of ensembles with feature selection applied independently for each module. However, in this work, we will not be concerned with the decision functions for each module. Instead, we will evaluate the intersection of the sets of false positives and false negatives produced by the four algorithms in each module. This will give us the optimal performance that each module could achieve.

The most important advantages of this model is the possibility to execute the algorithms in parallel and the modularity allowing the exchange of any algorithm of the ensemble without any modification of the rest. Nowadays, it is not rare to find multi-core processors in personal computers. Each algorithm could run on a different core to speed up the computation and make up for the computational overload introduced by the number of algorithms. A multi-core architecture would even allow the data preprocessing modules to work in parallel alleviating the time needed to analyse each packet.

4.3 Most Relevant Features

Table 4.5 shows the most important features for each class of attacks according to [50]. The features are selected using support vector machines (SVM), linear genetic programming (LGP) and multivariate adaptive regression splines (MARS). Features which are selected by different algorithms for the same class of attacks are highlighted because they should definitely be in the subset of features used to detect that class of attack.

Surprisingly, neither "protocol_type" nor "service" were selected by the three algorithms for the `DoS` class of attacks. Even if their experiments were conducted on a hierarchical SOM, Kayacik et al. [34] concluded that those features were the most significant for this class of attacks. See Appendix A for a description of the different features in the KDD99 dataset.

4.4 Description of the Experiments

The problem of intrusion is divided into five distinct subproblems:

- detection of `Probe` attacks
- detection of `U2R` attacks
- detection of `R2L` attacks
- detection of `DoS` attacks
- detection of `Normal` instances

Each problem is handled by one or more algorithms of the ensemble as mentioned above. This allows us to treat each sub-problem separately in the experiments and to join the sub-solutions into a general solution for the problem of intrusion detection. However, no algorithm was explicitly designed to detect `Normal` examples. In fact, if the number of algorithms is odd and the majority of the algorithms classify the example as `Normal`, then it will be classified by the ensemble as `Normal` as well. First, the dataset must be split into five files (one for each class of attacks plus one for the class `Normal`). The next step was to build a dataset for each sub-problem by sampling a number of examples in one class of attacks and the same number in the class `Normal` in order

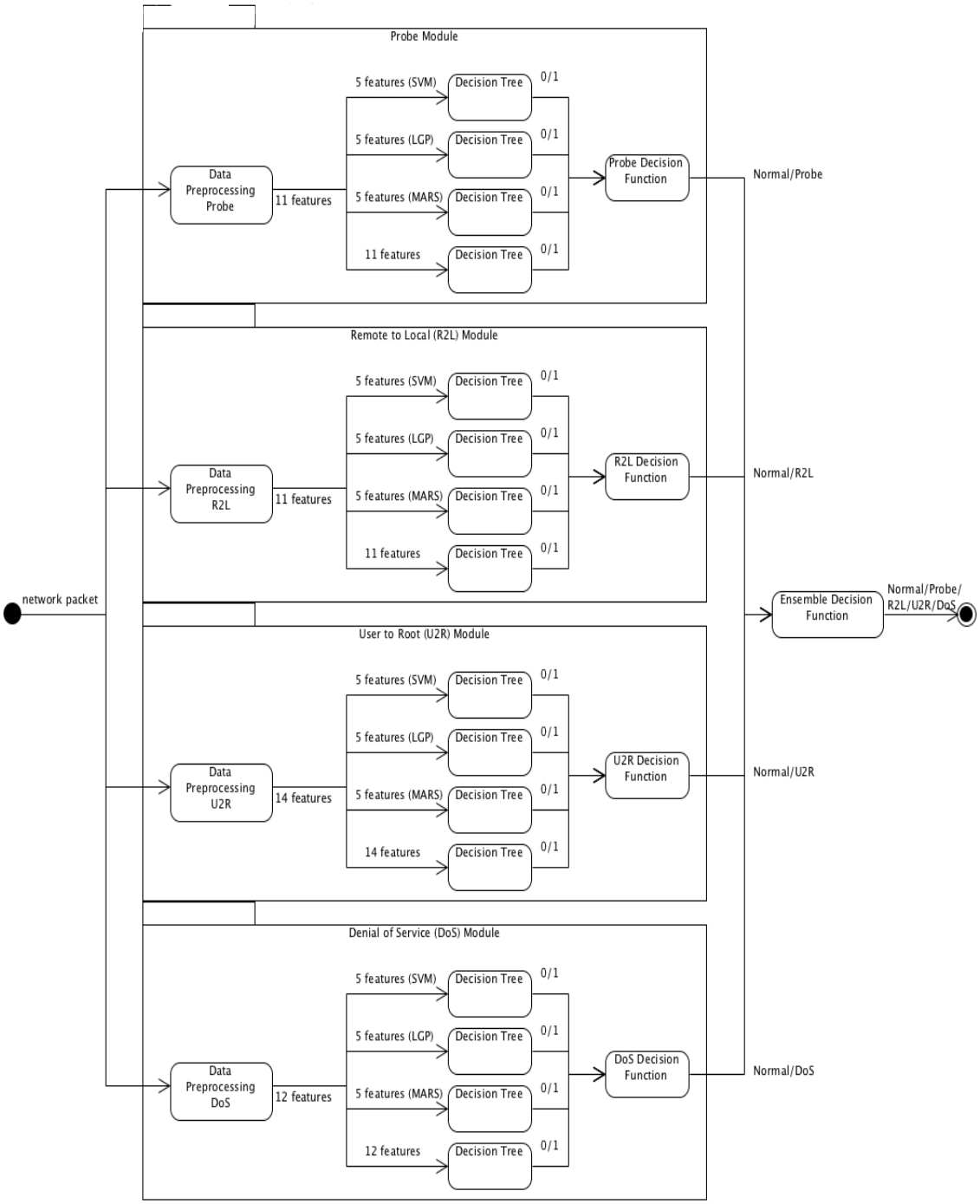


Figure 4.3: Model of the ensemble used in this thesis. Each algorithm is a binary classifier outputting 0 if the packet is considered normal traffic and 1 if the packet is classified as anomalous

Class of attacks	SVM features	LGP features	MARS features
Probes	src_bytes dst_host_srv_count count protocol_type srv_count	srv_diff_host_rate rerror_rate dst_host_diff_srv_rate logged_in service	src_bytes dst_host_srv_count dst_host_diff_srv_rate dst_host_same_srv_rate srv_count
User to Root	src_bytes duration protocol_type logged_in flag	root_shell dst_host_srv_serror_rate num_file_creations serror_rate dst_host_same_src_port_rate	dst_host_srv_count duration count srv_count dst_host_count
Remote to Local	srv_count service duration count dst_host_count	is_guest_login num_file_access dst_bytes num_failed_logins logged_in	srv_count service dst_host_srv_count count logged_in
Denial of Service	count srv_count dst_host_srv_serror_rate serror_rate dst_host_same_src_port_rate	count num_compromised wrong_fragments land logged_in	count srv_count dst_host_srv_diff_host_rate src_bytes dst_bytes
Normal	dst_bytes dst_host_count logged_in dst_host_same_srv_rate flag	dst_bytes src_bytes dst_host_rerror_rate num_compromised hot	dst_bytes src_bytes logged_in service hot
Nr of different features	**16 features**	**21 features**	**13 features**

Table 4.5: Most relevant features of the KDD99 dataset for each class of attacks according to [50]

to have a balanced dataset with 50% anomalous examples and 50% normal examples. A balanced dataset is necessary to avoid the problem of skewed classes where the accuracy of the predictor can be made artificially high by increasing the number of instances from one of the classes. An alternative configuration could include a much higher number of normal examples such as 90% or even 99.99% which would be more realistic compared to what is generally observed on a computer network. In this case, it is important to understand that the baseline would shift to 90% or 99.99%. The problem with this distribution is that the interpretation of the results is less obvious. For example, if the predictor obtains 99.99% accuracy, it means that the predictor is just guessing. Besides, what is the meaning of 99.999% accuracy?

For the classes of attacks with few examples such as `R2L` and `U2R`, the entire set is selected which leads to a `U2R` training set of 104 examples (52 `U2R` and 52 `Normal`) and a `R2L` training set of 2,252 examples (1,126 `R2L` and 1,126 `Normal`). Furthermore, the labels of the datasets for each class of attacks are slightly modified to allow binary classifiers such as SVM to process the data. The two labels are 0 for a `Normal` instance and 1 for an "anomalous" instance. It is always possible to change this setting to allow multiple class classification. This would improve the accuracy of the alert message delivered by the IDS to the network administrator. However, this modification would also add a computational overhead on the learning algorithms and is not needed for the purpose of our experiments. Finally, the equations used to measure the performance of the IDS are described in Section 2.1.2.

The experiments performed are in the direct continuity of the work done in [50, 51, 72]. In those papers, Srinivas Mukkamala and Andrew H. Sung identify the key features relevant to each of the four classes of attacks. The first step of the experiments is to assess the sets of features selected in [50]. Afterwards, another experiment will take place where an ensemble of machine learning algorithms will be fed with those sets.

The objectives of the experiments are multiple. In particular, the experiments have to answer the following questions:

- Can ensemble approaches improve the accuracy and speed of the detection even when using the simplest algorithms without fine tuning them?
- Are the results of [50], concerning the features selected by the three algorithms SVM, MARS and LGP for each class of attack, correct?
- Is the predictor fast enough to be used in a real-world application? (see Section 2.1.2 for more details)
- Are the false positive rate and the false negative rate close enough to zero in order for the IDS to be efficient? (see Section 2.1.1 for more details)

4.5 Development Tools

The experiments were performed using a "Intel Core 2 Duo, 2,26 GHz processor with 4 GB of RAM". The comparisons made with the 10 Gb/s Ethernet network in terms of speed are informative. It is obvious that a commercial NIDS will run on a much more competitive machine than the one used in these experiments.

The program used to develop the experiments is `Matlab R2012a`. The project can be cloned using the following command:

```
git clone git@git.assembla.com:ensemble-based-intrusion-detection.git
```

or simply followed at:

```
http://www.assembla.com/spaces/ensemble-based-intrusion-detection/
```

The main function is:

```
ensembleForIDS(istesting, class, trainingset, testset)
```

where

- `istesting` is a boolean value used to choose if the program should run only the training phase with cross-validation or should assess the performance of the ensemble on the test set.
- `class` is a string which can take one of the values in `['u2r', 'r2l', 'probe', 'dos']` and is useful only if `istesting` is false. In this case, the program will perform a cross-validation on the training set for the class mentioned in the parameter `class`.
- `trainingset` & `testset` are structures containing the training set and test set for each class of attacks. For example, the training set for `DoS` attacks can be accessed with the following command: `trainingset.dos`. Loading these two sets can take a few minutes on the machine described above. That is why a separate `Matlab` file, called `getData.m`, is in charge of loading them before the experiment takes place. This file should be loaded manually once at the beginning of the experimentation to ensure that the datasets are available throughout the experiments.

Additionally, `R` files are available to split the original training set and test set into five files for each class of attacks. However, these ten files are already available on the git repository mentioned above.

The commands used for the machine learning algorithms are:

- SVM
  - `options = optimset('MaxIter',max_iter);` specifies the maximum number of iterations allowed for the SVM.
  - Training: `model_svm = svmtrain(X.tr,Y.tr,'kernel_function','rbf','options',options);`
    - `X.tr` is a matrix of dimension $m \times n$ where m is the number of examples and n is the number of variables in the training set.
    - `Y.tr` is a vector of dimension $m \times 1$ where m is the number of examples. `Y.tr` represents the labels for each example in the training set. `Y.tr(i)` is the label corresponding to the *ith* example of `X.tr`.
    - `('kernel_function', 'rbf')` specifies that the kernel function used for the SVM is a Gaussian Radial Basis Function.
    - `('options', options)` specifies the options given above.
  - Prediction: `Ynew = svmclassify(model_svm,X.ts);`
    - `model_svm` represents the model built by `svmtrain` in the training phase.
    - `X.ts` is a matrix of dimension $m \times n$ where m is the number of examples and n is the number of variables in the test set.
- Decision Tree
  - Training: `ctree = ClassificationTree.fit(X.tr,Y.tr);`
    - `X.tr` is a matrix of dimension $m \times n$ where m is the number of examples and n is the number of variables in the training set.
    - `Y.tr` is a vector of dimension $m \times 1$ where m is the number of examples. `Y.tr` represents the labels for each examples in the training set. `Y.tr(i)` is the label corresponding to the *ith* example of $X.tr$.
  - Prediction: `Ynew = predict(ctree,X.ts);`
    - `ctree` represents the model built by `ClassificationTree.fit` in the training phase.
    - `X.ts` is a matrix of dimension $m \times n$ where m is the number of examples and n is the number of variables in the test set.

The `svmtrain` and `svmclassify` functions belong to the Bioinformatics Toolbox.
The `ClassificationTree.fit` and `predict` belong to the Statistic Toolbox.

4.6 Experiment 1: Feature Selection Assessment

In this experiment, several classifiers were trained with a different number of features. The algorithm used as a classifier is SVM with a Gaussian radial basis function kernel (RBF) which is one of the most powerful machine learning algorithms currently available. The maximum number of iterations is set to 10,000 to allow the algorithm to converge. The goal of the experiment is not concerned with finding the best algorithm possible and fine-tuning it. Instead, what matters is to conclude on how well the algorithm performs with a smaller set of features. In this case, it is only natural to use exactly the same setting for the algorithms and to compare the performance based only on the sets of features. Four SVMs were trained with four different sets of features. Only the training set was used for this experiment. The results obtained represent the performance of the algorithms on the cross-validation set which is extracted from the training set. Experiment 2 will assess the performance of the algorithms on the test set.

The first SVM is trained with all the 41 features available in the dataset. The three last are trained with five features selected in [50] by the three algorithms LGP, SVM and MARS for each class of attacks. These features are listed in Table 4.5 (on page 45). The results obtained in terms of accuracy were compared to those obtained in [58]. The term "problematic instances", used in Tables 4.8, 4.11, 4.15, 4.19 and 4.23, means that these instances were classified wrongly by all algorithms. In short, Experiment 1 aims to assess the performance of the algorithms on the training set for each class of attacks with different sets of features selected in [50] and discover which examples are problematic and why.

After a few trials, some changes were made. The first one was to switch from SVM to decision tree (DT) which is much faster. Moreover, SVM could have given an advantage to the features selected by the SVM used as feature selection algorithm in [50]. The second change was to add a new set of features. This set of features, that we called "combined", is the union of the sets of features selected by SVM, LGP and MARS from which redundant features have been removed. The number of features in each "combined" set is 11 for `Probe`, 14 for `U2R`, 11 for `R2L` and 12 for `DoS`. The reason for this additional sets is the fact that they help bringing down the number of FN and FP as we will see in the results of the experiments. On top of that, there is no additional cost from the extraction of these features from the original network packets since they have to be extracted for the other algorithms anyway.

Finally, for the FP and FN analysis, we call `ensemble_best` the number of examples wrongly classified by all three algorithms trained with the sets of five features and the algorithm trained by the "combined" set of features. This is the best result that an ensemble composed of these four algorithms could achieve if the combination of their individual results was optimal. These numbers are calculated by taking the intersection of the set of examples misclassified for each algorithm. The experiment was run ten times for each class of attacks to ensure accuracy of the results and to find the types of attack in each class that are misclassified most of the times by the `ensemble_best`. The values displayed in all tables, except the ones representing the number of problematic instances for the `ensemble_best` wrt the FP and FN for 10 runs of the algorithm, are the average values over the 10 validation sets of the 10-fold cross-validation.

4.6.1 Probe

4.6.1.1 Experimental Settings

The number of examples selected randomly in the `Probe` dataset is equal to 10,000. The same number of `Normal` instances are selected from the `Normal` dataset. In total, the training set used to train the algorithms to detect `Probe` attacks contains 20,000 entries. The number 10,000 is chosen to have a significant sample with as many different examples as possible without affecting too much the training time. A 10-fold cross-validation is used to assess the models. With the setting of the experiment, 2,000 (10%) entries are used as a validation set for each iteration of the 10-fold cross-validation.

4.6.1.2 Results

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.995600	0.998584	0.992613	0.995587	0.001802
SVM (5 features)	0.973250	0.967279	0.979699	0.973431	0.030060
LGP (5 features)	0.969300	0.960884	0.978470	0.969579	0.033937
MARS (5 features)	0.866650	0.872720	0.858602	0.865576	0.124212

Table 4.6: Results of Experiment 1 for the class `Probe`

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	37.383806	0.290233
SVM (5 features)	24.953225	0.391659
LGP (5 features)	26.408618	0.427812
MARS (5 features)	93.545649	1.480771

Table 4.7: Results of Experiment 1 for the class `Probe` with respect to the computational speed

Tables 4.6 and 4.7 show that the set of features selected by MARS obtains the worse results both in terms of accuracy and speed. The other sets perform better but are still quite far from the accuracy of the set of 41 features both in terms of speed and accuracy. With a testing time of 0.290233 seconds to classify 2,000 examples, this set of features would be able to handle only 6.891 examples/s on a machine similar to the one used for the experiment. This is way too slow for a real-time application performing at current network bandwidth. It can be noted, in Table 4.8, that there is a significant drop of FP and FN when looking at all algorithms. This is good news because it means that, only by selecting appropriate subsets of features to train the algorithms, all algorithms do not classify all instances in the same way. As expected, using several algorithms in an ensemble can improve significantly the FPR and false negative rate (FNR). Unfortunately, 7 false positives and 9 false negatives over 20,000 examples are still too high numbers to be useful in a real-world application. In the example of a 10Gb/s Ethernet network given in Section 2.1.1, this result would lead to a little bit less than 5,250 FP/s and 6,750 FN/s. Furthermore, a difficult point is to find an appropriate combination of the output of all algorithms. Based on the results, a majority vote with weights depending on the accuracy of each algorithm might be a good solution. In this case, since the predictors are binary classifiers, the number of algorithms taking part in the voting process should obviously be odd to avoid ties. Eventually, another interesting fact is that the default setting of SVM with a gaussian radial basis function obtains better results for the `Probe` class than the ones obtained by [58] when using a polynomial kernel.

	Nr of False Positives	Nr of False Negatives
41 features	18	62
Problematic instances	7	9

Table 4.8: Experiment 1 (`Probe`): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation

To ensure that the results did not give any advantage to the algorithm trained with the set of features selected by the SVM algorithm, the experiment was carried out again with a decision tree as a classifier. Tables 4.9, 4.10 and 4.11 show the results obtained with DT. The accuracy is exactly the same as [58] obtained with 41 features: 99.86% accuracy on this class with the same decision tree. As expected, the training and prediction speed are much faster than with SVM (see Table 4.10). With the worst prediction speed of 0.015412 seconds to classify 2,000 examples, the classifier is now able to handle 132,100 entries per second. This is a tremendous improvement even though it is still far from the 15,000,000 entries per second that are necessary to perform on a 10 Gb/s Ethernet network. Unfortunately, the difference between the prediction time when using all 41 features and when using only 5 features does not live up to our expectations. Although, even if the speed of the prediction is not improved very much by the feature selection, a smaller set of features means that less features must be extracted from the network packet. Achieving similar accuracy with less features is still a very good improvement. A less expected result is that the accuracy of all classifiers increases when using DT instead of SVM (see Table 4.9). In particular, the accuracy of the algorithm trained with the set of features selected by MARS jumps

from 0.866650 to 0.997500 which is a huge improvement. The other two algorithms trained with sets of 5 features are also closing in on the one trained with the set of 41 features. The algorithm fed with the five features selected by LGP performs slightly worse than the others with an F1score of 0.993211 and could be replaced by a more accurate algorithm.

Finally, the selected sets of features seem to be a good choice when the algorithms are trained using decision trees. Similarly to the case with SVM as a predictor, the overall numbers of false positives and false negatives drop significantly when using more than one algorithm (see Table 4.11). When the experiment is run ten times (see Table 4.12), the average number of FP is 0.7 and the average number of FN is 3 over 20,000 examples for the `ensemble_best`. All types of attacks appear at least once as an FN, however, "satan" and "portsweep" seem to be the most difficult attacks to detect as shown in Table 4.12. When comparing the problematic instances of "satan", "portsweep" and "ipsweep" with regular instances of these same types of attacks, it seems that "src_bytes" is the feature that gives the biggest trouble to the algorithm. In fact, for probe attacks, "src_bytes" should be very small if not equal to zero. Whenever an example of these attacks has a high value for "src_bytes", it goes undetected. This is a big problem because an attacker could easily fill the packets of the attack with random bytes to evade the IDS. We could think that it would be a good idea to get rid of this feature, however, "src_bytes" is a very important feature to detect `Probe` attacks because the only algorithm that performs poorly is the one trained with the set of features selected by LGP and this set of features does not include "src_bytes". In case of "satan", "logged_in" and "srv_diff_host_rate" are normally equal to zero. The sample of regular instances for each problematic attack type as well as the sample of problematic instances that helped drawing these conclusions can be found in Appendix B, Section B.1.

To conclude, even if the goal of the experiment was not to show the difference of performance between two types of algorithms, SVM and DT, it seems that the choice of SVM affects greatly the set of features selected by MARS. On top of that, DT outperforms SVM in terms of speed and accuracy for the sub-problem of `Probe` confirming the results obtained in [58]. The accuracy obtained by the `ensemble_best` is much better than what a single algorithm is capable of, but not good enough for a real-world application. Eventually, since DT totally outperformed SVM, the next experiments will be carried out using DT instead of SVM.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.998650	0.998516	0.998790	0.998652	0.001514
SVM (5 features)	0.998150	0.998400	0.997902	0.998150	0.001600
LGP (5 features)	0.993200	0.992322	0.994107	0.993211	0.007706
MARS (5 features)	0.997500	0.997782	0.997196	0.997488	0.002184
Combined	0.998950	0.999198	0.998714	0.998955	0.000799

Table 4.9: Results of Experiment 1 for the class `Probe` with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

4.6.2 User to Root

4.6.2.1 Experimental Settings

The results of these experiments are likely to be worse than with the `Probe` or `R2L` class because of the scarce number of data related to the `U2R` class. The `U2R` dataset contains 52 examples which will all be selected for this experiment. The same number of `Normal` instances are selected from the normal dataset. In total, the training set used to train the algorithms to detect `U2R` attacks contains 104 entries. A 10-fold cross-validation is used to assess the models. With the

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	1.008131	0.015412
SVM (5 features)	0.260827	0.014153
LGP (5 features)	0.279878	0.013080
MARS (5 features)	0.241558	0.014044
Combined	0.274144	0.012810

Table 4.10: Results of Experiment 1 for the class `Probe` with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	12	17
ensemble_best	0	3
Problematic instances	0	3

Table 4.11: Experiment 1 (`Probe` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation

Run nr	FP	FN	Attack names (Total nr of instances)			
			satan (15,892)	portsweep (10,413)	ipsweep (12,481)	nmap (2,316)
1	0	2	1	1	0	0
2	1	1	1	0	0	0
3	0	4	4	0	0	0
4	1	4	1	1	1	1
5	1	4	3	1	0	0
6	0	4	1	2	1	0
7	1	1	0	1	0	0
8	1	1	0	1	0	0
9	0	2	2	0	0	0
10	2	7	4	2	0	1
Average	0.7	3				
Max	2	7				

Table 4.12: Experiment 1 (`Probe` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation for 10 runs of the program for the `ensemble_best`. The total number of instances for each attack is given as an indication since only 10,000 examples are selected randomly from the entire `Probe` dataset. Consequently, the number of FN observed for each attack is not exactly out of the total number of instances.

setting of the experiment, 10 (∼10%) entries are used as a validation set for each iteration of the 10-fold cross-validation. The values displayed in all tables represent the average values over the 10 validation sets of the 10-fold cross-validation. As mentioned in the first experiment, the algorithm used as a classifier from now on is the decision tree.

4.6.2.2 Results

As expected, the results shown in Table 4.13 are worse than for `Probe`, but it is only natural since each FP and FN have a bigger impact on the general accuracy because of the small number of examples. These results are much better than in [58] who obtained 68.00% accuracy on this class with the same decision tree. In particular, the algorithm trained with the set of features selected by LGP perfoms poorly again. An interesting result is that the algorithms trained with a set of features selected by SVM and MARS perform better than the one trained on the set of all 41 features. This is probably caused by the small number of examples. In this case, 41 features are too many to generalize well. The training time is obviously smaller than for the other classes of attack, but the testing time is actually higher even though there are only 10 examples to predict. Table 4.14 shows the results linked to the running time of the algorithms. Finally, Table 4.15 displays very good results for the combination of the algorithms with smaller sets of features. No false positives and no false negatives are found when looking at the intersection of the FP and FN of the four algorithms. This seems like very good news; however, when running the experiment ten times, in general either one FP or one FN appears (see Table 4.16). The FP can be explained by the small number of examples in the dataset, only 52 `Normal` examples are present. The FN is always a "rootkit" attack which is wrongly classified as normal traffic, but it is not always the same instance. This indicates that some information is missing for the Decision Tree to classify "rootkit" attacks correctly. Moreover, there are only 10 "rootkit" attacks in the `U2R` dataset. Again, [39] does not document this kind of attack. However, McAfee and Symantec describe it in details in [46] and [13], respectively. It appears that "rootkit" can be any kind of malware such as worm, Trojan or virus with the ability to hide its presence and actions to the users and processes of a computer; this is called a stealth attack. Thanks to these two reports, we are now able to partially understand why "rootkit" attacks are so difficult to detect for our algorithms. The diversity found in malware has probably a huge impact on the problem and the fact that there are only 10 examples in the dataset increases the difficulty. The values taken by these 10 examples for the 14 features of the combined algorithm are shown in Appendix B, Section B.3. Almost all of these 10 instances have very different values for the 14 features. The `ensemble_best` performs perfectly in most cases, but it is difficult to conclude anything with such a small dataset. One FP or FN out of 10 instances of the cross-validation set is quite a bad score.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.930000	0.935714	0.930000	0.918772	0.078333
SVM (5 features)	0.960000	0.954762	0.980000	0.964413	0.070000
LGP (5 features)	0.900000	0.910714	0.868333	0.881269	0.078333
MARS (5 features)	0.970000	0.949048	1.000000	0.972106	0.061667
Combined	0.960000	0.945714	0.980000	0.961197	0.061667

Table 4.13: Results of Experiment 1 for the class `U2R` with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	0.062540	0.017726
SVM (5 features)	0.039750	0.016328
LGP (5 features)	0.043447	0.016070
MARS (5 features)	0.037551	0.015165
Combined	0.039882	0.015765

Table 4.14: Results of Experiment 1 for the class `U2R` with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	4	3
ensemble_best	0	0
Problematic instances	0	0

Table 4.15: Experiment 1 (`U2R` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation

Run nr	FP	FN	Attack names
1	0	1	rootkit
2	0	0	NONE
3	0	0	NONE
4	0	0	NONE
5	0	0	NONE
6	1	1	rootkit
7	0	0	NONE
8	0	0	NONE
9	0	0	NONE
10	2	1	rootkit
Average	0.3	0.3	
Max	2	1	

Table 4.16: Experiment 1 (`U2R` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation for 10 runs of the program for the `ensemble_best`

4.6.3 Remote to Local

4.6.3.1 Experimental Settings

The R2L dataset contains 1,126 examples which will all be selected for this experiment. The same number of Normal instances are selected from the normal dataset. In total, the training set used to train the algorithms to detect R2L attacks contains 2,252 entries. A 10-fold cross-validation is used to assess the models. With the setting of the experiment, 225 (∼10%) entries are used as a validation set for each iteration of the 10-fold cross-validation. As mentioned in the previous experiment, the algorithm used as a classifier from now on is the decision tree.

4.6.3.2 Results

The results shown in Table 4.17 are very similar to those obtained for the class Probe even though the number of instances in the dataset is much smaller. These results are also much better than in [58] who obtained 84.19% accuracy on this class with the same decision tree. This experiment clarify the fact that classifying Probe attacks and R2L attacks are two very distinct problems even if they are both intrusions and that is why they should be treated separately. Again, the selected features seem to be a good choice even if a little drop of accuracy can be observed compared to Probe. In particular, the algorithm trained with the set of 5 features selected by MARS has a high rate of false positives and the one trained with the set of features selected by LGP has the lowest accuracy but also a lower FPR which implies a higher false negative rate. The combination of all algorithms helps to bring down the number of false positives and false negatives, but these numbers are again too high for a real-world application. There are eight different types of R2L attacks represented in the R2L dataset. After running the experiments ten times, only three types of these attacks trigger false negatives for the ensemble_best: "spy", "imap" and "phf". The results are displayed in Table 4.20. There is not much documentation about "spy" attacks which are not even represented in the test set. However, the signatures of "imap" and "phf" are described in [39]. Detection of these attacks requires very specific features. In the case of a "phf" attack, the IDS "must monitor http requests watching for invocations of the phf command with arguments that specify commands to be run." [40]. None of the 41 features depicted in the KDD99 dataset gives any information about a specific command being run on the system. It would be impractical to do so for each specific command triggering an attack. However, this could be the reason behind the incapacity of the machine learning algorithm to detect these kind of attacks with certainty. Without meaningful information, the algorithm is powerless in building a proper model. There are two ways to solve this problem, either new features have to be added to the dataset or an IDS using signatures of attacks should perform the detection for these particular types of attacks. In the former case, the new features should not be too specific to ensure that new attacks could also be identified. In the second case, the IDS loses its ability to detect similar attacks but its accuracy increases. To detect an imap attack an IDS should be "programmed to monitor network traffic for oversized Imap authentication strings" [40]. This description seems more within reach of our IDS since "service" and "src_bytes" are both represented in the dataset. Nevertheless, most of the time, there are no FN for the R2L class, but too many FP as shown in Table 4.20. FP are more difficult to evaluate since DT was used as a binary classifier. In this case, it is not possible to know for which attack the normal instance was mistaken. However, it is probable that the kind of attacks triggering FN are the same that trigger FP. An n-class classifier with n representing the number of types of attacks in one class could help to have more insight into the problem. Since SVM was originally chosen as the classifier and is normally used as a binary classifier, the data was processed to accommodate this setting.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.990222	0.987983	0.992174	0.990019	0.011029
SVM (5 features)	0.985778	0.983745	0.987755	0.985693	0.015705
LGP (5 features)	0.973778	0.964643	0.983134	0.973667	0.007706
MARS (5 features)	0.980444	0.976529	0.984339	0.980260	0.022462
Combined	0.989333	0.989481	0.989231	0.989232	0.010683

Table 4.17: Results of Experiment 1 for the class `R2L` with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	0.190301	0.011375
SVM (5 features)	0.061773	0.011516
LGP (5 features)	0.049170	0.011439
MARS (5 features)	0.070769	0.011180
Combined	0.079923	0.011857

Table 4.18: Results of Experiment 1 for the class `R2L` with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	17	10
ensemble_best	4	1
Problematic instances	2	0

Table 4.19: Experiment 1 (`R2L` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation

Run nr	FP	FN	Attack names (Total nr of instances)		
			phf (4)	spy (2)	imap (12)
1	8	4	3	1	0
2	8	0	0	0	0
3	4	0	0	0	0
4	4	0	0	0	0
5	8	1	0	0	1
6	8	0	0	0	0
7	4	0	0	0	0
8	6	0	0	0	0
9	8	0	0	0	0
10	8	0	0	0	0
Average	6.6	0.5			
Max	8	4			

Table 4.20: Experiment 1 (`R2L` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation for 10 runs of the program for the `ensemble_best`

4.6.4 Denial of Service

4.6.4.1 Experimental Settings

The `DoS` dataset contains 3,883,370 examples. "neptune" and "smurf" attacks count for the major part of the examples in the `DoS` training set with 1,072,017 and 2,807,886 examples respectively. The other types of attacks have much smaller number of examples as shown in Table 4.1. For example, the type of `DoS` called "land" is represented only 21 times out of almost 4 millions entries. For this reason, a sample of 5,000 examples have be selected randomly from the "neptune" set and another 5,000 was selected randomly from the "smurf" set. All examples of the other types of attacks have been included leading to a `DoS` training set of 13,467 examples. The same number of `Normal` instances was selected from the normal dataset leading to a total of 26,934 examples to train the algorithm. A 10-fold cross-validation is used to assess the models. With the setting of the experiment, 2,693 (∼10%) entries are used as a validation set for each iteration of the 10-fold cross-validation. The values displayed in all tables represent the average values over the 10 validation sets of the 10-fold cross-validation. As mentioned in the first experiment, the algorithm used as a classifier is the decision tree.

4.6.4.2 Results

The accuracies of all algorithms are displayed in Table 4.21. Again, the results are better on this class with the same decision tree than in [58] who obtained 96.83% accuracy. In this experiment, the algorithm trained with the set of features selected by SVM obtains the worse score with an F1score equal to 0.933876 whereas the set of features selected by MARS gets the best score 0.998621 after the set of all features and the set of combined features. This is important because it means that there is a set of 5 features that can perform almost as well as the set of 41 features even when the number of training example is not scarce. The improvement in term of speed can be observed in Table 4.22. The testing time drops from 0.014081 s to 0.009620 s. This is not a huge difference, but every bit counts when designing a real-time application. The fact that a smaller set of features has to be extracted from the network packets and that the algorithm is capable of a high accuracy with limited features is a important improvement in itself. Another good result is the fact that the `ensemble_best` achieved an average number of FP equal to zero after running the program 10 times (see Table 4.24). Table 4.23 shows that the number of FN is reduced as well. Three types of attacks trigger FN: "smurf", "neptune" and "back". The first two types rarely appear in the list. However, the most difficult type of attack to handle seems to be "back". This is not a surprise, since to detect a "back" the IDS must look for a big number of frontslashes ("/") in the request URL [39]. There are no features in the dataset that take this particularity into account. Consequently, the model has to rely on other features to make up for this lack of information and this leads to an imperfect result. Nevertheless, as expected, the `ensemble_best` brings robustness to the accuracy of the IDS.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.999480	0.999552	0.999405	0.999478	0.000443
SVM (5 features)	0.933457	0.927760	0.940103	0.933876	0.073239
LGP (5 features)	0.986892	0.988686	0.985076	0.986872	0.011296
MARS (5 features)	0.998626	0.999104	0.998138	0.998621	0.000887
Combined	0.999257	0.999479	0.999032	0.999255	0.000518

Table 4.21: Results of Experiment 1 for the class `DoS` with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	1.283539	0.014081
SVM (5 features)	0.592327	0.011113
LGP (5 features)	0.388289	0.009812
MARS (5 features)	0.214126	0.009620
Combined	0.487517	0.012633

Table 4.22: Results of Experiment 1 for the class `DoS` with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	6	8
ensemble_best	0	2
Problematic instances	0	2

Table 4.23: Experiment 1 (`DoS` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation

Run nr	FP	FN	Attack names (Total nr of instances)		
			smurf (5,000)	back (2,203)	neptune (5,000)
1	0	2	1	1	0
2	0	2	1	1	0
3	0	2	1	1	0
4	0	1	0	1	0
5	0	2	0	2	0
6	0	1	0	1	0
7	0	2	0	2	0
8	0	1	0	1	0
9	0	2	0	1	1
10	0	1	0	1	0
Average	0	1.6			
Max	0	2			

Table 4.24: Experiment 1 (`DoS` - Decision Tree): Sum of the number of problematic instances wrt the FP and FN of each validation set in the 10-fold cross-validation for 10 runs of the program for the `ensemble_best`

4.6.5 Discussion

Experiment 1 has shown that ensemble approach is indeed a very powerful paradigm that can be used to bring down the number of FP and FN. The lower accuracy observed by individual algorithms is countered by the union of their results. Even with sets containing only five features, the results are very encouraging. Moreover, treating each class of attack as a different problem solved by a specialised algorithm seems to work well when compared to strategies using one algorithm to detect all classes of attacks. "Divide and conquer" and "Unity is strength" seem to be opposite views, but they are actually both applied in this work with impressive results. In general, algorithms using fewer features have slightly lower accuracy and prediction time but much lower training time. The results obtained by [50] seem to be correct. However, the set of features selected by LGP gives the worst result in most cases except for `DoS` where it is the set of features selected by SVM which performs poorly. Consequently, the sets of features selected by LGP should be reconsidered for all classes except `DoS` and the set of features selected by SVM should be replaced in the case of `DoS`.

The number of different types of attacks that go undetected is very small and only few examples of these attacks are problematic. Most of the time, the problem lays in the lack of information contained in the dataset. Some attacks require very specific features and should probably be handled by specialized programs or signature-based IDSs. The class `Probe` is a bigger problem since most of the attacks belonging to this class exploit a legitimate feature used by network administrators. As a result, all types of `Probe` attacks trigger FN at some point even though "portsweep" and "satan" are the most problematic.

Since one of the goals of the experiment was to show that the prediction time can be lowered greatly by using smaller sets of features, we can say that it is not the case. However, as previously mentioned, a smaller set means that only a few features must be extracted from the network packet in the data preprocessing phase. Since the accuracy is not lowered too much in the best cases, this is a huge improvement that could be used in real IDSs. If these algorithms are used in parallel, the detection speed would be greatly increased.

Moreover, the union of all algorithms using fewer features improves tremendously the accuracy. In particular, in average over 10 runs of the program, only 0.7 FP and 3 FN are observed for the `Probe` class over 20,000 examples, 6.6 FP and 0.5 FN for the `R2L` class over 2,252 examples, 0.3 FP and 0.3 FN for the `U2R` class over 104 examples and 0 FP and 1.6 FN for the `DoS` class over 20,000 examples. Even though these results are much better than what could be achieved with a single algorithm, they are still quite far from being useful in a real-world application where these numbers should be lower than 1 for almost 15 millions examples in a 10Gb/s Ethernet network. However, we can argue that within the 15 millions examples 90% will be `Normal` traffic containing no attack at all. Still, the `ensemble_best` has to be improved further to stand a chance against clever hackers. Moreover, the results described above are the best that an ensemble composed of these algorithms and sets of features could achieve. In its current state, there is no point in building an experiment to assess a real combination of the results of the individual algorithms in the `ensemble_best`. Further work will have to be carried out to find the best suitable algorithms and sets of features. Nevertheless, it is interesting to see how well this `ensemble_best` can perform when predicting previously unseen attack types. That is the topic of the second experiment.

4.7 Experiment 2: Model Assessment of the Test Set

In this experiment, several classifiers were trained with different number of features on examples from the training set. The algorithm used as a classifier is decision tree (DT). The goal of the experiment is to evaluate the same model used in the previous experiment on the test set after training it on the training set. As we have seen in Section 4.1, Table 4.2 shows that the test set is

composed of many examples with unseen attacks (attacks that are not represented in the training set). This experiment aims to assess if the ensemble is capable of generalizing to new types of attacks belonging to the same classes as the ones previously seen in the training set. The values represented in all tables represent one run of the program. The number of examples selected for the training set in each class is the same as in the first experiment.

4.7.1 Probe

4.7.1.1 Experimental Settings

The `Probe` test set contains 4,166 examples which are all selected for this experiment. The same number of `Normal` instances are selected from the normal test set leading to a total of 8,332 examples to test the algorithm.

4.7.1.2 Results

The accuracy of all algorithms degraded drastically in comparison to the first experiment as shown in Table 4.25. In particular, the set of features selected by SVM obtains the worst results with an F1score of 0.713054 whereas the set selected by LGP manages to keep a respectable accuracy with an F1score of 0.857766. Generally speaking, the testing time only doubled even though the number of examples to classify increased fourfold (see Table 4.26). The `ensemble_best` still seems to work properly on the test set, but the number of FN is very high. This observation can be seen in Tables 4.27 and 4.27. Table 4.27 shows that the `ensemble_best` is able to handle a part of the new attacks, but does not recognize them as easily as the old ones. The most surprising fact is that the attack "ipsweep" seems to go undetected almost all the time. This result is very unusual because "ipsweep" was available in the training set and did not cause any trouble in the previous experiment. One possible explanation is that the examples of "ipsweep" from the test set are very different from the ones in the training set. After examining the training set carefully, typical values for the features of an "ipsweep" attack can be observed and are shown in Table 4.29. When compared to the values of "ipsweep" in the test set, it appears that these values are exactly in the range described in Table 4.29 refuting our hypothesis.

To conclude, the results are not as bad as they look. First, almost all old attacks are perfectly detected, especially "portsweep" and "satan" which triggered FN in the first experiment are now absent from the attacks triggering FN. The new attacks are detected most of the time, but the number of FN is still too high to be useful in a real-world application. Finally, solving the problem of "ipsweep" would bring down tremendously the number of FN.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.930869	0.977140	0.882381	0.927346	0.020643
SVM (5 features)	0.776284	0.993991	0.555929	0.713054	0.003361
LGP (5 features)	0.874820	0.993053	0.754921	0.857766	0.005281
MARS (5 features)	0.840374	0.964309	0.706913	0.815789	0.026164
Combined	0.799688	0.762787	0.869899	0.812829	0.270523

Table 4.25: Results of Experiment 2 for the class `Probe` with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	1.008573	0.026632
SVM (5 features)	0.222444	0.035693
LGP (5 features)	0.216063	0.028129
MARS (5 features)	0.195481	0.025189
Combined	0.297489	0.026340

Table 4.26: Results of Experiment 2 for the class `Probe` with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	86	490
ensemble_best	6	363
Problematic instances	6	286

Table 4.27: Experiment 2 (`Probe` - Decision Tree): number of problematic instances wrt the FP and FN

Run nr	FP	FN	Attack names (Total nr of instances)		
			ipsweep (306)	mscan (1,053)	saint (736)
1	6	363	255	106	2
2	36	581	306	174	101
3	2	426	303	20	103
4	5	411	306	4	101
5	8	837	303	433	101
Average	11.4	523.6			
Max	8	837			

Table 4.28: Experiment 2 (`Probe` - Decision Tree): Sum of the number of problematic instances for the `ensemble_best` wrt the FP and FN for 5 runs of the program

count	dst_host_diff_srv_rate	service	dst_host_srv_count	logged_in	protocol_type
1	0	12	1-255	0	1
rerror_rate	dst_host_same_srv_rate	src_bytes	srv_diff_host_rate	srv_count	
0	1	8 or 18	1-50	1 or 0	

Table 4.29: Experiment 2 (`Probe` - Decision Tree): Typical values for “ipsweep” attack

4.7.2 User to Root

4.7.2.1 Experimental Settings

The `U2R` test set contains 70 examples which are all selected for this experiment. The same number of `Normal` instances are selected from the normal test set leading to a total of 140 examples to test the algorithm. Three new attack types have been added to the test set: "ps", "xterm" and "sqlattack".

4.7.2.2 Results

Table 4.30 shows results very similar to those in Experiment 1 except for the set of features selected by SVM which performs poorly with an F1score of 0.562500. The set selected by SVM does not seem to generalize well to new types of attacks. The best algorithm is the one trained with the "combined" set of features outperforming even the algorithm trained with all 41 features in the same way that was observed in Experiment 1. The `ensemble_best` brings down the number of FP to 1 and the number of FN to 0 with an average value of 1.6 and 1 respectively over five runs of the program (see Tables 4.32 and 4.33). As expected, sometimes a "rootkit" attack goes undetected as was the case in Experiment 1. Besides, "ps" appears also rarely as an FN. The most surprising result comes from undetected "buffer overflow" even though it never happened in the previous experiment. However, "xterm" and "sqlattack" are detected all the time which is good because it means that the `ensemble_best` generalizes well for the `U2R` class.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.900000	0.951613	0.842857	0.893939	0.042857
SVM (5 features)	0.400000	0.442623	0.771429	0.562500	0.971429
LGP (5 features)	0.835714	0.943396	0.714286	0.813008	0.042857
MARS (5 features)	0.850000	0.945455	0.742857	0.832000	0.042857
Combined	0.942857	0.942857	0.942857	0.942857	0.057143

Table 4.30: Results of Experiment 2 for the class `U2R` with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	0.058578	0.020856
SVM (5 features)	0.050880	0.018908
LGP (5 features)	0.046631	0.019206
MARS (5 features)	0.027604	0.014730
Combined	0.029518	0.014111

Table 4.31: Results of Experiment 2 for the class `U2R` with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	3	11
ensemble_best	1	0
Problematic instances	0	0

Table 4.32: Experiment 2 (`U2R` - Decision Tree): number of problematic instances wrt the FP and FN

Run nr	FP	FN	Attack names (Total nr of instances)		
			ps (16)	rootkit (13)	buffer_overflow (22)
1	1	0	0	0	0
2	2	0	0	0	0
3	1	3	1	2	0
4	0	0	0	0	0
5	4	2	0	0	2
Average	1.6	1			
Max	4	3			

Table 4.33: Experiment 2 (`U2R` - Decision Tree): Sum of the number of problematic instances for the `ensemble_best` wrt the FP and FN for 5 runs of the program

4.7.3 Remote to Local

4.7.3.1 Experimental Settings

The `R2L` test set contains 16,347 examples which are all selected for this experiment. The same number of `Normal` instances are selected from the normal test set leading to a total of 32,694 examples to test the algorithm. We can expect very bad results because of the poor distribution of attacks in the `R2L` training set. In fact, most of the attacks are "warezclient" (1020 out of 1126 in total for the `R2L` training set) leaving only 106 instances of all other attack types (seven different types) to train the algorithms. Moreover, "warezclient" is not even represented in the test set. There is no chance that the models built by the different algorithms will perform well on new attacks or even on old ones with this limited training set.

4.7.3.2 Results

Table 4.34 shows the already expected results. The accuracy of all algorithms is equal or close to 50% which leaves place to a guessing game. The only set of features which stands out slightly is the one selected by LGP, but with an FPR of 0.559919 it is not really worth mentioning. The number of FN obviously explodes as displayed in Tables 4.36 and 4.37. Old and new types of attacks are similarly misclassified. There is nothing really interesting to be observed from these results. The only conclusion that can be drawn is that the `R2L` training set contains too little examples of each types of attack to be of any help.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.500000	NaN	0.000000	NaN	0.000000
SVM (5 features)	0.500000	NaN	0.000000	NaN	0.000000
LGP (5 features)	0.610326	0.582303	0.780571	0.667015	0.559919
MARS (5 features)	0.500000	NaN	0.000000	NaN	0.000000
Combined	0.500000	NaN	0.000000	NaN	0.000000

Table 4.34: Results of Experiment 2 for the class `R2L` with Decision Tree

4.7.4 Denial of Service

4.7.4.1 Experimental Settings

The same procedure that was performed with the training set has to be applied again with the test set. The `DoS` test set contains 229,853 examples. "neptune" and "smurf" attacks count for

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	0.279557	0.066044
SVM (5 features)	0.095524	0.040405
LGP (5 features)	0.068128	0.045206
MARS (5 features)	0.105602	0.037055
Combined	0.083530	0.035023

Table 4.35: Results of Experiment 2 for the class R2L with respect to the computational speed (Decision Tree)

	Nr of False Positives	Nr of False Negatives
41 features	0	16,347
ensemble_best	1	3,587
Problematic instances	1	3,587

Table 4.36: Experiment 2 (R2L - Decision Tree): number of problematic instances wrt the FP and FN

			Attack names (Total nr of instances)												
Run nr	FP	FN	ftp_write (3)	guess_passwd (4,367)	httptunnel (158)	multihop (18)	named (17)	xsnoop (4)	phf (2)	sendmail (17)	snmpgetattack (7,741)	snmpguess (2,406)	warezmaster (1,602)	worm (2)	xlock (9)
1	1	3,587	2	11	146	12	14	4	2	9	8	2,312	1,066	2	4
2	1	7,778	1	3,609	145	10	12	0	1	6	522	2,406	1,066	0	0
3	1	15,610	1	4,221	145	11	12	0	1	6	7,741	2,406	1,066	0	0
4	1	8,340	2	3,618	146	12	14	4	2	9	522	2,406	1,600	2	3
5	1	3,578	2	11	145	9	12	3	1	3	8	2,312	1,066	2	4
Average	1	7,778.6													
Max	1	15,610													

Table 4.37: Experiment 2 (R2L - Decision Tree): Sum of the number of problematic instances for the ensemble_best wrt the FP and FN for 5 runs of the program

the major part of the examples in the DoS test set with 58,001 and 164,091 examples respectively. The other types of attacks have much smaller number of examples as shown in Table 4.4. For example, the type of DoS called "land" is represented only 9 times and "udpstorm" is represented only twice. For this reason, a sample of 5,000 examples was selected randomly from the "neptune" set and another 5,000 was selected randomly from the "smurf" set. All examples of the other types of attacks have been included leading to a DoS test set of 17,761 examples. The same number of Normal instances are selected from the normal test set leading to a total of 35,522 examples to test the algorithm.

4.7.4.2 Results

Results, shown in Table 4.38, are much worse than in the first experiment. For instance, the set of features selected by LGP obtain by far the worst results with an F1score of 0.695487. Nevertheless, all other algorithms perform better than the one trained with all features. The set selected by SVM is the best with an F1score of 0.879408. Moreover, the major part of FN can be assigned to new attacks. The "pod" attack is the only old attack that triggers a few FN for each run of the program (see Table 4.41). Other old attacks sometimes triggering FN include "smurf" and "neptune", but the number of FN for these attacks is very low. New attacks are more problematic. In particular, Table 4.41 shows that "mailbomb", "apache2", "processtable" and "udpstorm" recurrently trigger FN even if a large portion of these attacks are detected in general. The ensemble_best helps bringing down the FP from 69 to 21 and the FN from 7,268 to 459. This is quite an improvement, but again is not enough for a real-world application. In conclusion, we can say that the ensemble_best performed quite well on unseen DoS attacks, but that its generalization power is still limited.

Feature selection algorithm	Accuracy	Precision	Recall	F1score	FPR
41 features	0.793452	0.993467	0.590789	0.740953	0.003885
SVM (5 features)	0.876978	0.862369	0.897134	0.879408	0.143179
LGP (5 features)	0.761049	0.958469	0.545746	0.695487	0.023647
MARS (5 features)	0.821998	0.995495	0.646923	0.784220	0.002928
Combined	0.853612	0.995269	0.710602	0.829183	0.003378

Table 4.38: Results of Experiment 2 for the class DoS with Decision Tree. The line marked in red highlights the set of features that obtained the worst performance for this class of attack.

Feature selection algorithm	Training time (in sec)	Cross-validation time (in sec)
41 features	1.516960	0.048521
SVM (5 features)	0.638737	0.039207
LGP (5 features)	0.423280	0.036156
MARS (5 features)	0.235077	0.033683
Combined	0.513214	0.037918

Table 4.39: Results of Experiment 2 for the class DoS with respect to the computational speed (Decision Tree)

4.7.5 Discussion

The goal of this experiment was to find out if the ensemble_best could generalize well on new unseen examples of attacks. Even if in general the ensemble_best helps tremendously to bring

	Nr of False Positives	Nr of False Negatives
41 features	69	7,268
ensemble_best	21	459
Problematic instances	21	452

Table 4.40: Experiment 2 (DoS - Decision Tree): number of problematic instances wrt the FP and FN

			Attack names (Total nr of instances)						
Run nr	**FP**	**FN**	mailbomb (5,000)	apache2 (794)	neptune (5,000)	pod (87)	processtable (759)	smurf (5,000)	udpstorm (2)
1	21	459	366	15	0	2	74	0	2
2	18	1,002	366	220	0	6	407	1	2
3	15	631	207	345	0	2	75	0	2
4	17	427	0	66	1	6	348	4	2
5	12	922	367	64	0	6	481	2	2
Average	16.6	688.2							
Max	21	1,002							

Table 4.41: Experiment 2 (DoS - Decision Tree): Sum of the number of problematic instances for the ensemble_best wrt the FP and FN for 5 runs of the program

down the number of FP and FN, it is still far from reaching the accuracy appropriate to a real-world application. In particular, datasets which are not careful designed are proved to be useless in building accurate models of the attacks. This is the case with the R2L training set which contains mainly examples of the "warez_client" attack which is not even represented in the test set and very few examples of all other types of attacks. The performance of the ensemble_best was acceptable for the classes of attacks U2R and DoS. The performance on the Probe class was also standard even though "ipsweep" attacks went undetected for unknown reasons. Overall, we can say that the results of this second experiment were not very satisfying, but once again proved the usefulness of the ensemble approach.

In the future, particular attention has to be paid to the features relevant to each attack. New features carrying meaningful information about the attacks must be designed to help the machine learning algorithms to successfully classify all types of attack. DoS and Probe classes are mostly characterized by time-related features whereas R2L and U2R classes mostly are characterized by content-related features extracted from the payload of the network packets. This can only be achieved if security experts and machine learning experts work hand in hand towards this goal.

Chapter 5

Concluding Remarks

5.1 Conclusions

The aim of this thesis was to show that ensemble approaches fed with appropriate features sets can help tremendously in reducing both the number of false positives and false negatives. In particular, our work showed that the sets of relevant features are different for each class of attacks and that is why it is important to treat those classes separately. Based on several works concerning ensemble approaches applied to intrusion detection systems, we developed our own IDS to evaluate the relevance of the sets of features selected in [50]. The system built in this thesis was in fact an ensemble of four ensembles of decision trees. Each of these four ensembles was in charge of detecting one class of attacks and was composed of four decision trees trained with different sets of features. The first three decision trees were fed with sets of five features selected by Mukkamala et al. in [50]. The last decision tree was fed with the union of these three sets of five features from which the redundant features were removed.

The experiments showed that these sets were appropriate in most cases. In experiment 1, the set of features selected by linear genetic programming gave the worst results, except for the class `DoS` for which the set of features selected by SVM performed poorly. Experiment 2 gave less interesting results because of the inappropriate distribution of examples between the training set and test set of the KDD99 dataset. In particular, the ensemble could not generalize properly on the `R2L` class because the training set contains mainly the type of attacks "warez_client" which is not even represented in the test set. In both experiments, we looked at the number of instances that were misclassified by all four algorithms in order to obtain a result from the best combination of these algorithms. Further work would be required to develop a real decision function combining the results of the different algorithms. However, since the accuracy obtained in this work was not good enough for a real-world application, designing decision functions was unnecessary. Nevertheless, we are convinced that this work was heading towards the right direction in order to overcome the limitations of current intrusion detection systems.

Finally, a thorough analysis of the examples that were misclassified by the ensemble was also performed. This analysis helped us understand why the algorithms were unable to classify properly those instances. In particular, the types of attacks that were systematically misclassified by the ensemble were highlighted, and by looking at the signature of these attacks, we were able to find out the reason behind those errors of classification. In most cases, the attacks displayed very specific features that were not captured by the set of variables in the dataset. These attacks should probably be handled by a specialized system or else new variables should be developed to train the machine learning algorithms. In all cases, we believe that security experts and machine learning experts should work together with the aim of improving the current intrusion detection systems. On the one hand, security experts should develop additional features needed to detect network

attacks and should share their knowledge on attack mechanisms with machine learning experts. On the other hand, Machine learning experts should share their knowledge about the learning algorithms and explain, for example, why an algorithm would perform better on a particular class of attacks. Understanding only one aspect of the problem is definitely not enough to overcome the difficult challenges of intrusion detection systems.

The main limitation that must be overcome in order to build efficient intrusion detection systems is the lack of labelled datasets. Since new attacks are developed constantly, it is difficult and expensive to create an up-to-date dataset that would include all kinds of attacks on a network. Moreover, the information displayed in the dataset must be anonymized to avoid privacy violations. For these reasons, unsupervised learning could be a more appropriate solution to the problem of intrusion detection. Active learning is also a candidate to solve the problem as we will discuss in Section 5.3.

5.2 Research Contributions

- This thesis provides a survey of the state-of-the-art in the field of ensemble approaches applied to intrusion detection systems.
- Additionally, this work has shown that each class of attacks should be treated separately. In fact, at least one algorithm should be assigned to detect one class of attacks instead of using a single algorithm to detect all classes of attacks. Furthermore, algorithms used to detect different classes of attacks should be trained with different sets of features.
- The experiments have also concluded that ensembles fed with different sets of features for each class of attacks can outperform more standard approaches even when the ensemble is composed of several simple classifiers such as decision trees.

5.3 Future Work

Many improvements can be added to the intrusion detection system developed in this thesis. Since the list is quite long, we have decided to focus on the most important ones which are described in more details in this section. Further ideas for improvement can be found in the list of reference books on the topic of IDS at [64] and in the compilation of papers on IDS at [65] and on IPS at [66].

Ensemble approaches using bagging techniques are very modular systems as discussed throughout this thesis. A framework should be developed for ensemble approaches applied to intrusion detection. This would facilitate the replacement of any algorithm of the ensemble and ease the addition of more algorithms to each class of attacks. An object oriented architecture exploiting heritage and polymorphism could be a good candidate for this kind of framework. Different machine learning algorithms could be added as modules to the framework and be represented in the program by an object initialized at run time. For example, all algorithms could inherit from the class Predictor. The pseudo-code shown in Algorithm 1 summarizes the idea. The number of algorithms as well as the type of each algorithm for each class of attacks could be determined on a user interface in an initialization step. Additionally, the framework should exploit multicore processors in order to increase the speed of the computation. With the help of the previously described framework, it would be easy to experiment many combinations of algorithms for each class of attacks in order to build the most accurate system.

The next improvement would transform our system into an IPS. In real-world applications, it is important for the system not only to detect the intrusion, but also to react to the detection in order to stop the attack or avoid similar attacks in the future. The IPS must be able to protect the

Algorithm 1: Ensemble framework

```
Predictor predictor;
for i ← 1 to nb_DT do
    predictor[i] ← new DT(...);
end
for j ← i to i + nb_SVM do
    predictor[j] ← new SVM(...);
end
...;
for k ← 0 to predictor.size do
    predictor[k].train(...);
end
```

system without external help of an administrator. This kind of reactive systems can be difficult to develop in practice as we have seen in Section 2.1.4. For this reason, more research must be carried out in order to overcome these limitations.

Another improvement would be to apply a signature-based detection system with a database of signatures of limited size first to filter known attacks. If an attack is detected in this first step, a response module should be triggered to stop the attack. If no attack is detected, the data would be transferred to the ensemble described in this thesis. Finally, if an attack is detected by the ensemble, a signature generator could be triggered to update the signature database. This kind of system is described in more details in Section 2.1.7.

Since the lack of labelled dataset is a key problem to the evolution of intrusion detection systems, it would be a good idea to look at machine learning mechanisms such as active learning to overcome this challenge. Active learning is a semi-supervised machine learning technique in which the algorithm tries to build a model of the data with limited labels. When examples lays at the boundary of the model, the algorithm asks the help of a human expert to label those specific examples. Consequently, since only a small portion of the dataset needs to be labelled, the cost of this tedious task is greatly reduced. There is very little literature on active learning applied to the problem of intrusion detection even though this technique could be a very serious candidate to solve the dataset problem in this field.

Bibliography

[1] *Cost-Based Modeling for Fraud and Intrusion Detection: Results from the JAM Project*, volume 2, Hilton Head, South Carolina, January 2000. IEEE Computer Society.

[2] Ajith Abraham and Crina Grosan. *Evolving Intrusion Detection Systems*, volume 13 of *Studies in Computational Intelligence*, pages 57–79. Springer-Verlag, Berlin, Heidelberg, 2006.

[3] Ajith Abraham, Crina Grosan, and Carlos Martin-vide. Evolutionary Design of Intrusion Detection Programs. *International Journal of Network Security*, 4(3):328–339, November 2006.

[4] Ajith Abraham and Ravi Jain. Soft Computing Models for Network Intrusion Detection Systems. In *Classification and Clustering for Knowledge Discovery*, volume 4, pages 191–207. Springer, 2005.

[5] Ajith Abraham, Ravi Jain, Johnson P. Thomas, and Sang-Yong Han. D-SCIDS: Distributed soft computing intrusion detection system. *J. Network and Computer Applications*, 30(1):81–98, January 2007.

[6] Ajith Abraham and Johnson Thomas. *Distributed Intrusion Detection Systems: A Computational Intelligence Approach*, volume 5, pages 105–135. Idea Group Inc. Publishers, 2005.

[7] Abdulrahman Alharby and Hideki Imai. Hybrid intrusion detection model based on ordered sequences. In *Proceedings of the 3rd international conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, MMM-ACNS'05, pages 352–365, St. Petersburg, Russia, September 2005. Springer-Verlag.

[8] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, Fort Washington, Pennsylvania, 1980.

[9] Emna Bahri, Nouria Harbi, and Hoa Nguyen Huu. Approach Based Ensemble Methods for Better and Faster Intrusion Detection. In *Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems*, Lecture Notes in Computer Science, pages 17–24, Torremolinos-Malaga, Spain, June 2011. Springer.

[10] Ryan Barnett, Josh Zlatin, Brian Rectanus, and Roberto Salgado. OWASP ModSecurity Core Rule Set Project, 2009. https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project. Webpage. Last accessed date: 19th of June 2012. Last update: June, 2012.

[11] Markus Brameier. *On linear genetic programming*. PhD thesis, Universität Dortmund, 2005.

[12] Bulba and Kil3r. Bypassing Stackguard and Stackshield. *Phrack Magazine*, 10(56), May 2000. http://www.phrack.org/issues.html?issue=56&id=5.

[13] James Butler and Sherri Sparks. Windows rootkits of 2005, part one. Technical report, Symantec, Cupertino, CA 95014 USA, 2010. Last update:02 Nov 2010. Last accessed date: 10 Jul 2012.

[14] James Cannady. Applying CMAC-Based On-Line Learning to Intrusion Detection. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*, volume 5, pages 405–410, Como, Italy, July 2000. IEEE.

[15] CERT. Cyber Security Engineering (CSE) team CERT : Statistics (Historical), 2012. http://www.cert.org/stats/. Webpage. Last accessed date: 4th of April 2012. Last update: February, 2009.

[16] CERT Coordination Center staff. CERT® Advisory CA-1996-01 UDP Port Denial-of-Service Attack, 1996. http://www.cert.org/advisories/CA-1996-01.html. Webpage. Last accessed date: 1st of June 2012. Last update: September, 1997.

[17] Patrikakis Charalampos, Michalis Masikos, and Olga Zouraraki. Distributed Denial of Service Attacks. *The Internet Protocol Journal*, 7(4):13–35, December 2004. http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-4/dos_attacks.html.

[18] Srilatha Chebrolu, Ajith Abraham, and Johnson P. Thomas. Hybrid Feature Selection for Modeling Intrusion Detection Systems. In *Proceedings of the 11th International Conference on Neural Information Processing*, volume 3316 of *Lecture Notes in Computer Science*, pages 1020–1025, Calcutta, India, November 2004. Springer-Verlag.

[19] Srilatha Chebrolu, Ajith Abraham, and Johnson P. Thomas. Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, 24(4):295–307, June 2005.

[20] Cheng-Yuan Ho, Yuan-Cheng Lai, I-Wei Chen, Fu-Yu Wang, and Wei-Hsuan Tai. Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems. *Communications Magazine, IEEE*, 50(3):146–154, March 2012.

[21] Crispin Cowan, Calton Pu, Dave Maier, Heather Hintony, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th conference on USENIX Security Symposium*, SSYM'98, pages 63–78, San Antonio, Texas, January 1998. USENIX Association.

[22] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. *Buffer overflows: attacks and defenses for the vulnerability of the decade*, pages 227–237. IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[23] Jonathan J. Davis and Andrew J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security*, 30(6-7):353–375, June 2011.

[24] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31:805–822, April 1999.

[25] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[26] Rohit Dhamankar, Mike Dausin, Marc Eisenbarth, and James King. The Top Cyber Security Risks - Zero-Day Vulnerability Trends, 2009. http://www.sans.org/top-cyber-security-risks/zero-day.php. Webpage. Last accessed date: 12th of June 2012. Last update: September, 2009.

[27] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. GP Ensemble for Distributed Intrusion Detection Systems. In *Proceedings of the 3rd International Conference on Advances in Pattern Recognition (ICAPR)*, pages 54–62, Bath, UK, August 2005.

[28] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. An ensemble-based evolutionary framework for coping with distributed intrusion detection. *Genetic Programming and Evolvable Machines*, 11:131–146, June 2010.

[29] Luca Foschini, Ashish V. Thapliyal, Lorenzo Cavallaro, Christopher Kruegel, and Giovanni Vigna. A Parallel Architecture for Stateful, High-Speed Intrusion Detection. In *Proceedings of the 4th International Conference on Information Systems Security*, ICISS '08, pages 203–220, Hyderabad, India, December 2008. Springer-Verlag.

[30] Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1–67, June 1991.

[31] Silvia González, Javier Sedano, Álvaro Herrero, Bruno Baruque, and Emilio Corchado. Testing ensembles for intrusion detection: On the identification of mutated network scans. In *Proceedings of the 4th international conference on Computational intelligence in security for information systems*, CISIS'11, pages 109–117, Torremolinos-Malaga, Spain, June 2011. Springer-Verlag.

[32] Mohamed G. Gouda and Xiang-Yang Alex Liu. Firewall design: consistency, completeness, and compactness. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 320–327, Hachioji, Tokyo, March 2004.

[33] D.M. Gregg, W.J. Blackert, D.C. Furnanage, and D.V. Heinbuch. Denial of service (DOS) attack assessment analysis report. Technical report, Johns Hopkins University, Baltimore, Maryland, 2001.

[34] H. Gunes Kayacik, A. Nur Zincir-Heywood, and Malcolm I. Heywood. A hierarchical SOM-based intrusion detection system. *Engineering Applications of Artificial Intelligence*, 20(4):439–451, June 2007.

[35] Isabelle Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[36] Lars Kai Hansen and Peter Salamon. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.

[37] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. *Security and Privacy, IEEE Symposium on*, 0:296–305, May 1990.

[38] Paul Innella. The Evolution of Intrusion Detection Systems. Technical report, Tetrad Digital Integrity, LLC, Washington, United States, 2001.

[39] Kristopher Kendall. A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Master's thesis, Massachusetts Institute of Technology, 1999.

[40] Kristopher Kendall. Intrusion Detection Attacks Database, 2007. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/docs/attackDB.html. Webpage. Last accessed date: 24th of June 2012.

[41] Constantinos Kolias, Georgios Kambourakis, and M. Maragoudakis. Swarm intelligence in intrusion detection: A survey. *Computers & Security*, 30(8):625–642, September 2011.

[42] John R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[43] Christopher Kruegel, Fredrik Valeur, and Giovanni Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*, volume 14 of *Advances in Information Security*. Springer-Verlag, 2005.

[44] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning intrusion detection: supervised or unsupervised? In *Proceedings of the 13th ICIAP conference on image analysis and processing*, pages 50–57. Springer, September 2005.

[45] Felix Lau and Stuart H. Rubin. Distributed Denial of Service Attacks. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 2275–2280, Sheraton Music City Hotel, Nashville, Tennessee, USA, October 2000.

[46] McAfee. Rootkits, Part1 of 3: The Growing Threat. Technical report, McAfee, Inc., Santa Clara, CA 95054 USA, 2006. Last update:02 Nov 2010. Last accessed date: 10 Jul 2012.

[47] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Software Security Series. Addison-Wesley Professional, February 2006.

[48] Jim McMillan. Intrusion Detection FAQ: What is the difference between an IPS and a Web Application Firewall?, 2009. https://www.sans.org/security-resources/idfaq/ips-web-app-firewall.php. Website. Last accessed date: 22nd of June 2012. Last update: November, 2009.

[49] Peter Mell, Karen Scarfone, and Sasha Romanosky. A Complete Guide to the Common Vulnerability Scoring System Version 2.0, 2007. http://www.first.org/cvss/cvss-guide.html. Webpage. Last accessed date: 4th of April 2012.

[50] Srinivas Mukkamala, Andrew Sung, and Ajith Abraham. *Cyber Security Challenges: Designing Efficient Intrusion Detection Systems and Antivirus Tools*, pages 125–161. CRC Press, 2005.

[51] Srinivas Mukkamala and Andrew H. Sung. Identifying Significant Features for Network Forensic Analysis Using Artificial Intelligent Techniques. *International Journal of Digital Evidence*, 1(4):1–17, January 2003.

[52] Srinivas Mukkamala, Andrew H. Sung, and Ajith Abraham. Intrusion detection using an ensemble of intelligent paradigms. *Journal of Network and Computer Applications - Special issue on computational intelligence on the internet*, 28(2):167–182, April 2005.

[53] Andrew Ng. Machine Learning - Lecture 1 - Introduction, 2012. https://www.coursera.org/course/ml. Webpage. Last accessed date: 26th of June 2012. Last update: September, 2009.

[54] Andrew Ng. Machine Learning - Lecture 10 - Deciding what to try next, 2012. https://www.coursera.org/course/ml. Webpage. Last accessed date: 26th of June 2012. Last update: September, 2009.

[55] Andrew Ng. Machine Learning - Lecture 12 - Support Vector Machines, 2012. https://www.coursera.org/course/ml. Webpage. Last accessed date: 26th of June 2012. Last update: September, 2009.

[56] NVD. National Vulnerability Database : CVE and CCE Statistics Query Page, 2012. http://web.nvd.nist.gov/view/vuln/statistics-results?cves=on&query=&cwe_id=&pub_date_start_month=-1&pub_date_start_year=2000&pub_date_end_month=-1&pub_date_end_year=2012&mod_date_start_month=-1&mod_date_start_year=2000&mod_date_end_month=-1&mod_date_end_year=2012&cvss_sev_base=&cvss_av=&cvss_ac=&cvss_au=&cvss_c=&cvss_i=&cvss_a=. Webpage. Last accessed date: 4th of April 2012.

[57] Aleph One. Smashing The Stack For Fun And Profit. *Phrack*, 7(49), November 1996.

[58] Sandhya Peddabachigari, Ajith Abraham, Crina Grosan, and Johnson Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114–132, 2005.

[59] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Calgary, Alberta, Canada, 1998.

[60] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[61] Costin Raiu. The Top-10 of 2011: An "Explosive" Year in Security. Technical report, Kaspersky Lab, Voluntari, Ilfov, România., January 2012.

[62] Brandon Craig Rhodes, James A Mahaffey, and James D Cannady. Multiple Self-Organizing Maps for Intrusion Detection. In *Proceedings of the 23rd national information systems security conference*, pages 16–19, Baltimore MA. USA., October 2000. Citeseer.

[63] Matthew Richard. Intrusion Detection FAQ: Are there limitations of Intrusion Signatures?, 2001. https://www.sans.org/security-resources/idfaq/limitations.php. Webpage. Last accessed date: 15th of June 2012. Last update: April, 2001.

[64] SANS. Intrusion Detection FAQ: Books on or related to Intrusion Detection and Prevention. https://www.sans.org/security-resources/idfaq/reference_books.php. Webpage. Last accessed date: 22nd of June 2012.

[65] SANS. SANS InfoSec Reading Room - Intrusion Detection. https://www.sans.org/reading_room/whitepapers/detection/. Webpage. Last accessed date: 22nd of June 2012. Last update: June, 2012.

[66] SANS. SANS InfoSec Reading Room - Intrusion Prevention. https://www.sans.org/reading_room/whitepapers/intrusion/. Webpage. Last accessed date: 22nd of June 2012. Last update: June, 2012.

[67] Suseela T. Sarasamma, Qiuming A. Zhu, and Julie Huff. Hierarchical Kohonenen net for anomaly detection in network security. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(2):302–312, April 2005.

[68] Robert E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227, July 1990.

[69] Gregg Schudel. Bandwidth, Packets Per Second, and Other Network Performance Metrics, 2008. http://www.cisco.com/web/about/security/intelligence/network_performance_metrics.html. Webpage. Last accessed date: 20th of June 2012.

[70] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 305–316, Oakland, California, May 2010. IEEE Computer Society.

[71] Trustwave SpiderLabs. ModSecurity Open Source Web Application Firewall. http://www.modsecurity.org/projects/. Webpage. Last accessed date: 10th of June 2012.

[72] Andrew H. Sung and Srinivas Mukkamala. The Feature Selection and Intrusion Detection Problems. In *Proceedings of the 9th Asian Computing Science conference on Advances in Computer Science*, ASIAN'04, pages 468–482, Chiang Mai, Thailand, December 2004. Springer-Verlag.

[73] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the 2nd IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 53–58, Ottawa, Ontario, Canada, July 2009. IEEE Press.

[74] TippingPoint Zero Day Initiative researchers. Upcoming advisories, 2005. http://www.zerodayinitiative.com/advisories/upcoming/. Webpage. Last accessed date: 12th of June 2012. Last update: March, 2012.

[75] Ryan Trost. *Practical Intrusion Analysis: Prevention and Detection for the Twenty-First Century*. Addison-Wesley Professional, 1st edition, 2009.

[76] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. Intrusion detection by machine learning: A review. *Expert Systems with Applications: An International Journal*, 36(10):11994–12000, December 2009.

[77] Giovanni Vigna. Network Intrusion Detection: Dead or Alive? In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Austin, Texas, December 2010.

[78] Giovanni Vigna. Network Security Analysis. http://www.cs.ucsb.edu/~vigna/courses/cs279/. Webpage. Last accessed date: June 17, 2012, 2011.

[79] Timothy Wickham. Intrusion Detection Is Dead. Long Live Intrusion Prevention!, 2003. https://www.sans.org/reading_room/whitepapers/detection/intrusion-detection-dead-long-live-intrusion-prevention_1028. Webpage. Last accessed date: 14th of May 2012.

[80] Shelly Xiaonan Wu and Wolfgang Banzhaf. The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1–35, January 2010.

[81] Anazida Zainal, Mohd Aizaini Maarof, Siti Mariyam Shamsuddin, and Ajith Abraham. Ensemble of One-Class Classifiers for Network Intrusion Detection System. In *Proccedings of the 4th International Conference on Information Assurance and Security (IAS)*, IAS '08, pages 180–185, Napoli, Italy, September 2008. IEEE Computer Society.

[82] Peng Zhang, Xingquan Zhu, Yong Shi, Li Guo, and Xindong Wu. Robust ensemble learning for mining noisy data streams. *Decision Support Systems*, 50(2):469–479, January 2011.

Appendices

Appendix A

KDD99 dataset Features

The following tables describing the features of the KDD99 dataset were adapted from [1].

nr	Feature	description	Type
01	duration	Duration of the connection	Continuous
02	protocol type	Connection protocol (e.g. tcp,udp)	Symbolic
03	service	Destination service (e.g. telnet,ftp)	Symbolic
04	flag	Status flag of the connection	Symbolic
05	source bytes	Bytes sent from source to destination	Continuous
06	destination bytes	Bytes sent from destination to source	Continuous
07	land	1 if connection is from/to the same host/port; 0 otherwise	Symbolic
08	wrong fragment	number of wrong fragments	Continuous
09	urgent	number of urgent packets	Continuous

Table A.1: Basic features of individual TCP Connections

nr	Feature	description	Type
10	hot	number of “hot” indicators	Continuous
11	num failed logins	number of failed logins	Continuous
12	logged in	1 if successfully logged in; 0 otherwise	Symbolic
13	num compromised	number of “compromised” conditions	Continuous
14	root shell	1 if root shell is obtained; 0 otherwise	Symbolic
15	su attempted	1 if “su root” command attempted; 0 otherwise	Symbolic
16	num root	number of “root” accesses	Continuous
17	num file creations	number of file creation operations	Continuous
18	num shells	number of shell prompts	Continuous
19	num access files	number of operations on access control files	Continuous
20	num outbound cmds	number of outbound commands in a ftp session	Continuous
21	is hot login	1 if the login belongs to the “hot” list; 0 otherwise	Symbolic
22	is guest login	1 if the login is a “guest” login; 0 otherwise	Symbolic

Table A.2: Content features within a connection suggested by domain knowledge

nr	Feature	description	Type
23	count	number of connections to the same host as the current connection in the past two seconds	Continuous
24	srv count	number of connections to the same service as the current connection in the past two seconds	Continuous
25	serror rate	% of connections that have "SYN" errors (same-host connections)	Continuous
26	srv serror rate	% of connections that have "SYN" errors (same-service connections)	Continuous
27	rerror rate	% of connections that have "REJ" errors (same-host connections)	Continuous
28	srv rerror rate	% of connections that have "REJ" errors (same-service connections)	Continuous
29	same srv rate	% of connections to the same service	Continuous
30	diff srv rate	% of connections to different services	Continuous
31	srv diff host rate	% of connections to different hosts	Continuous
32	dst host count	count of connections having the same destination host	Continuous
33	dst host srv count	count of connections having the same destination host and using the same service	Continuous
34	dst host same srv rate	% of connections having the same destination host and using the same service	Continuous
35	dst host diff srv rate	% of different services on the current host	Continuous
36	dst host same src port rate	% of connections to the current host having the same src port	Continuous
37	dst host srv diff host rate	% of connections to the same service coming from different hosts	Continuous
38	dst host serror rate	% of connections to the current host that have an S0 error	Continuous
39	dst host srv serror rate	% of connections to the current host and specified service that have an S0 error	Continuous
40	dst host rerror rate	% of connections to the current host that have an RST error	Continuous
41	dst host srv rerror rate	% of connections to the current host and specified service that have an RST error	Continuous

Table A.3: Traffic features computed using a two-second time window

Appendix B

Problematic Instances

B.1 Probe

B.2 R2L

B.3 U2R

count	dst_host_diff_srv_rate	dst_host_same_srv_rate	dst_host_srv_count	logged_in	protocol_type	rerror_rate	service	src_bytes	srv_count	srv_diff_host_rate	label
PROBLEMATIC SATAN											
1	0.11	0.02	2	0	2	0	17	103	1	0	'satan.'
1	0.11	0.02	2	0	2	0	17	103	1	0	'satan.'
1	0.11	0.02	2	0	2	0	17	103	1	0	'satan.'
1	0.03	0.02	5	1	2	0	53	6	1	0	'satan.'
2	0.31	0.02	5	0	2	0	16	9	3	0.67	'satan.'
1	0.75	0.25	1	0	1	0	64	37	1	0	'satan.'
2	0.57	0.29	2	1	2	0	2	10	1	0	'satan.'
1	0.38	0.35	127	1	2	0	53	1710	1	0	'satan.'
1	0.39	0.34	124	1	2	0	53	1710	2	1	'satan.'
SAMPLE SATAN											
135	0.05	0.62	157	0	3	0	48	1	115	0	'satan.'
136	0.05	0.62	158	0	3	0	48	1	116	0	'satan.'
137	0.05	0.62	159	0	3	0	48	1	117	0	'satan.'
138	0.05	0.63	160	0	3	0	48	1	118	0	'satan.'
139	0.05	0.63	161	0	3	0	48	1	119	0	'satan.'
140	0.05	0.64	162	0	3	0	48	1	120	0	'satan.'
141	0.05	0.64	163	0	3	0	48	1	121	0	'satan.'
142	0.05	0.64	164	0	3	0	48	1	122	0	'satan.'
143	0.05	0.65	165	0	3	0	48	1	123	0	'satan.'
125	0.04	0.72	183	0	3	0	48	1	115	0	'satan.'
1	0.5	0.25	1	0	1	0	13	20	1	0	'satan.'
1	0.6	0.2	1	1	2	0	2	10	1	0	'satan.'
1	0.67	0.17	1	0	2	0	43	0	1	0	'satan.'
2	0.57	0.29	2	1	2	0	2	10	1	0	'satan.'
1	0.07	0.01	1	0	3	0	48	1	1	0	'satan.'
2	0.07	0.03	2	0	3	0	48	1	2	0	'satan.'
3	0.07	0.04	3	0	3	0	48	1	3	0	'satan.'
4	0.07	0.06	4	0	3	0	48	1	4	0	'satan.'
5	0.07	0.07	5	0	3	0	48	1	5	0	'satan.'
6	0.07	0.08	6	0	3	0	48	1	6	0	'satan.'
7	0.07	0.09	7	0	3	0	48	1	7	0	'satan.'
8	0.07	0.11	8	0	3	0	48	1	8	0	'satan.'
369	1	0	1	0	2	0.91	43	0	1	0	'satan.'
381	1	0	1	0	2	0.9	43	0	1	0	'satan.'
382	1	0	1	0	2	0.9	43	0	1	0	'satan.'
383	1	0	1	0	2	0.9	43	0	1	0	'satan.'
384	1	0	1	0	2	0.9	43	0	1	0	'satan.'
385	1	0	1	0	2	0.9	43	0	1	0	'satan.'
386	1	0	1	0	2	0.9	43	0	1	0	'satan.'
387	1	0	1	0	2	0.9	43	0	1	0	'satan.'
388	1	0	1	0	2	0.9	43	0	1	0	'satan.'
389	1	0	1	0	2	0.9	43	0	1	0	'satan.'
390	1	0	1	0	2	0.9	43	0	1	0	'satan.'

Figure B.1: Problematic satan

count	dst_host_diff_srv_rate	dst_host_same_srv_rate	dst_host_srv_count	logged_in	protocol_type	rerror_rate	service	src_bytes	srv_count	srv_diff_host_rate	label
PROBLEMATIC PORTSWEEP											
1	0.5	0	1	1	2	0	53	1273	1	0	'portsweep.'
1	0.47	0.04	10	1	2	0	53	1180	1	0	'portsweep.'
1	0.47	0.04	10	1	2	0	53	1180	1	0	'portsweep.'
src_bytes(=0)											
SAMPLE PORTSWEEP											
1	0.02	0.27	62	0	2	1	18	0	1	0	'portsweep.'
2	0.02	0.27	62	0	2	1	18	0	2	0	'portsweep.'
15	0.06	0.01	1	0	2	1	7	0	1	0	'portsweep.'
16	0.06	0.01	2	0	2	1	7	0	2	0	'portsweep.'
17	0.06	0.01	1	0	2	1	48	0	1	0	'portsweep.'
18	0.06	0.01	2	0	2	1	48	0	2	0	'portsweep.'
19	0.07	0.01	1	0	2	1	55	0	1	0	'portsweep.'
20	0.07	0.01	1	0	2	1	11	0	1	0	'portsweep.'
29	0.09	0.01	2	0	2	1	48	0	2	0	'portsweep.'
30	0.09	0	1	0	2	1	48	0	1	0	'portsweep.'
31	0.09	0.01	2	0	2	1	48	0	2	0	'portsweep.'
32	0.1	0	1	0	2	1	48	0	1	0	'portsweep.'
33	0.1	0.01	2	0	2	1	48	0	2	0	'portsweep.'
34	0.1	0.26	54	0	2	1	59	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	1	0	1	0	2	1	48	0	1	0	'portsweep.'
1	0.82	0	1	0	2	1	48	0	1	0	'portsweep.'
2	0.82	0.01	2	0	2	1	48	1	2	0	'portsweep.'
1	0.82	0	1	0	2	1	48	0	1	0	'portsweep.'
2	0.82	0.01	2	0	2	1	48	1	2	0	'portsweep.'
1	0.82	0	1	0	2	1	48	0	1	0	'portsweep.'
2	0.82	0.01	2	0	2	1	48	1	2	0	'portsweep.'
1	0.82	0	1	0	2	1	48	0	1	0	'portsweep.'

Figure B.2: Problematic portsweep

count	dst_host_diff_srv_rate	dst_host_same_srv_rate	dst_host_srv_count	logged_in	protocol_type	rerror_rate	service	src_bytes	srv_count	srv_diff_host_rate	label
PROBLEMATIC IPSWEEP											
1	0.99	0.03	3	1	2	0	59	4113	1	0	'ipsweep.'
1	0.99	0.03	3	1	2	0	59	4113	1	0	'ipsweep.'
1	0	1	80	0	2	1	18	0	1	0	'ipsweep.'
src_bytes(< 20)											
SAMPLE IPSWEEP											
1	0	1	1	0	2	1	18	0	1	0	'ipsweep.'
1	1	0.5	1	0	2	1	17	0	1	0	'ipsweep.'
1	1	0.33	1	1	2	1	55	0	1	0	'ipsweep.'
1	1	0.25	2	0	2	1	59	0	1	0	'ipsweep.'
1	1	0.2	1	0	2	1	48	0	1	0	'ipsweep.'
1	1	0.17	1	1	2	0	53	0	1	0	'ipsweep.'
1	1	0.03	1	0	2	0	9	0	1	0	'ipsweep.'
1	1	0.03	1	0	2	1	48	0	1	0	'ipsweep.'
1	1	0.03	1	0	2	1	48	0	1	0	'ipsweep.'
1	1	0.03	1	0	2	1	48	0	1	0	'ipsweep.'
1	0	1	255	0	1	0	12	8	27	1	'ipsweep.'
1	0	1	255	0	1	0	12	8	27	1	'ipsweep.'
1	0	1	255	0	1	0	12	8	28	1	'ipsweep.'
1	0	1	255	0	1	0	12	8	26	1	'ipsweep.'
1	0	1	255	0	1	0	12	8	29	1	'ipsweep.'
1	0	1	255	0	1	0	12	8	25	1	'ipsweep.'
1	0	1	255	0	1	0	12	8	30	1	'ipsweep.'
1	0	1	1	0	1	0	12	8	45	1	'ipsweep.'
1	0	1	2	0	1	0	12	8	8	1	'ipsweep.'
1	0	1	3	0	1	0	12	8	14	1	'ipsweep.'
1	0	1	4	0	1	0	12	8	15	1	'ipsweep.'
1	0	1	5	0	1	0	12	8	46	1	'ipsweep.'
1	0	1	6	0	1	0	12	8	7	1	'ipsweep.'
1	0	1	7	0	1	0	12	8	47	1	'ipsweep.'
1	0	1	8	0	1	0	12	8	6	1	'ipsweep.'
1	0	1	9	0	1	0	12	8	48	1	'ipsweep.'
1	0	1	10	0	1	0	12	8	5	1	'ipsweep.'
1	0	1	11	0	1	0	12	8	49	1	'ipsweep.'
1	0	1	12	0	1	0	12	8	4	1	'ipsweep.'
1	0	1	87	0	1	0	12	18	1	0	'ipsweep.'
1	0	1	88	0	1	0	12	18	1	0	'ipsweep.'
1	0	1	89	0	1	0	12	18	1	0	'ipsweep.'
1	0	1	90	0	1	0	12	18	1	0	'ipsweep.'
1	0	1	91	0	1	0	12	18	1	0	'ipsweep.'
1	0	1	92	0	1	0	12	18	1	0	'ipsweep.'

Figure B.3: Problematic ipsweep

1 count	2 dst_bytes	3 dst_host_count	4 dst_host_srv_count	5 duration	6 is_guest_login	7 logged_in	8 num_access_files	9 num_failed_logins	10 service	11 srv_count	12 label
1	10036	255	1	31	0	1	0	0	26	1	'imap.'
1	0	1	1	0	0	0	0	0	26	160	'imap.'
2	0	2	2	0	0	0	0	0	26	161	'imap.'
3	0	3	3	0	0	0	0	0	26	162	'imap.'
4	0	4	4	0	0	0	0	0	26	163	'imap.'
1	162	5	5	41	0	0	0	0	26	1	'imap.'
1	0	6	6	0	0	0	0	0	26	1	'imap.'
1	0	7	7	0	0	0	0	0	26	122	'imap.'
2	0	8	8	0	0	0	0	0	26	123	'imap.'
3	0	9	9	0	0	0	0	0	26	124	'imap.'
4	0	10	10	0	0	0	0	0	26	125	'imap.'
1	649186	11	11	0	0	0	0	0	26	1	'imap.'

Figure B.4: Problematic imap

1	2	3	4	5	6	7	8	9	10	11	12	13	14
count	dst_host_count	dst_host_same_src_port_rate	dst_host_srv_count	dst_host_srv_serror_rate	duration	flag	logged_in	num_file_creations	protocol_type	root_shell	serror_rate	src_bytes	srv_count
1	255	0	1	0	60	10	0	0	2	0	0	86	1
1	255	0	1	0	0	10	0	0	3	0	0	32	1
1	255	0	3	0	708	10	1	0	2	0	0	1727	1
1	1	1	41	0	0	10	1	0	2	0	0	0	1
1	255	0	4	0	98	10	1	1	2	1	0	621	1
1	255	0	1	0	21	10	1	1	2	0	0	89	1
1	255	0	2	0	60	10	0	0	2	0	0	90	1
1	255	0	4	0.2500	61	10	1	1	2	1	0	294	1
2	2	0.5000	2	0	0	10	0	0	3	0	0	4	2
1	1	1	1	0	0	10	0	0	3	0	0	4	1

Figure B.5: Rootkit attacks