Improving Intrusion Detection Systems through Machine Learning

Sebastiaan Tesink

ILK Research Group

Technical Report Series no. 07-02

Tilburg University

March 2007

Summary

An intrusion detection system (IDS) is a device, typically a designated computer system, that monitors activity to identify malicious or suspicious alerts. It is placed inside an organisation to monitor what occurs within the network of the organisation. The goal of an intrusion detection system is to accurately detect computer security incidents, and notify network administrators. A distinction is made between alerts and incidents by an intrusion detection system. Alerts are defined as all the observable actions on the computer network that are picked up by the sensors of an intrusion detection system. Incidents are malicious or suspicious alerts that have a high enough value to be considered a security-relevant system event in which the system's security policy is disobeyed or otherwise breached.

An IDS consists of four components, according to the Common Intrusion Detection Framework (CIDF); event generators, analysers, event databases and response units. In the research of this thesis, honeypots are used as event generators, providing alerts to the analyser. An effort will be made to choose a machine learning method that can be used as an analyser, which distinguishes alerts from incidents. An event database will be used to train the analyser, and to evaluate its predictions. The response units will not be within the scope of this thesis, but can be controlled by the decisions of the analyser.

The field of machine learning is concerned with the question of how to construct computer programs that automatically learn with experience. Machine learning will be used for the analyser to separate incidents from alerts, for three reasons. First, there is a large database that contains implicit regularities that are difficult for human beings to find. Second, it is hard to program an IDS using ordinary procedural programming languages, in a constantly changing environment. Third, the behaviour of an IDS is highly dependent on its environment.

Supervised machine learning is applied in this thesis, which means that the analyser will be trained using labelled alerts (either as incident or as a non-incident alert). These labels were provided by the Computer Emergency Response Team (CERT) of Tilburg University, and were used to predict labels of new unseen data. These labels were predicted in three ways; by eager learning, lazy learning and by hybrids.

In eager learning, most of the effort is invested in the learning phase, during which the machine learner abstracts beyond the training data. One disadvantage of this approach is that it generalises over individual instances, and thereby tends to forget instances. The form of eager learning used in this thesis is rule induction. In contrast to eager learning, lazy learning invests most of its effort in the classification task. One of the most common lazy learning algorithms is k-NN classification, which tests the similarity between a memory instance and a new instance. The implementation of k-NN used in this thesis is IB1.

Combining the efforts in the learning phase of an eager learning algorithm, and the efforts during the classification task of a lazy learner leads to hybrid machine learning. The complex rules produced by the eager learning are used to create binary feature vectors. These feature vectors can be used during the classification task, either as a replacement of the original features (Rules-R-H), or as extra features appended to the original feature set (Rules-A-H).

Lazy learners, eager learners and hybrid learners are evaluated for the use in a personalised intrusion detection system. During these evaluations, rule induction (a form of eager learning) seems to be highly effective, with the RIPPER implementation. With a precision of 83.10% and a recall of 99.48%, RIPPER outperforms both lazy learning and hybrid machine learning for this particular classification task. Another advantage is that RIPPER extracts knowledge from training sets that is intelligible to humans in the form of if-then rules.

Preface

While writing my thesis for Information Management[1], my interest in information security became more profound. On the other hand, during lectures of my second studies of Language and Artificial Intelligence, machine learning was one of the most challenging subjects. Combining both fields of interest leads to the obvious choice of applying machine learning on information security.

I hereby would like to thank my supervisor Antal van den Bosch, who gave me lots of inspiring ideas to write this thesis, with his views and conversations on this matter. Secondly, I would like to thank Bertjan Busser, who has helped me a lot, both providing me computational power on his servers, as well as his (very) quick assistance when I messed up while programming. I'm also very grateful to Iris Hendrickx, providing me scripts for the hybrid machine learners, and help on how to use them.

Without the help of UvT-CERT, this thesis could not have been finished. Thanks to Kees Leune, providing necessary information for my research, Teun Nijssen for answering privacy questions, and all the other UvT-CERT members for giving me the ability to do research in the field of information security.

One of the things I have learned during my research is that machine learning requires a lot of computational power. I owe Benedikt Kratz a game of tennis for providing me the computational facilities of the InfoLab, even after power-failures. Next to the computational power that the InfoLab provided, it also gave me a place to work and colleagues. I would like to thank Roel Kuijpers, for the many conversations and the great time we had, while writing our theses.

In order to come to a well-structured thesis, one needs someone to bother constantly with one's ideas, thus evaluating current ideas and coming to new ones. Many thanks to Jurrit de Vries for listening to me, and giving me useful feedback. I would also thank him for taking care of our company while I was busy writing my thesis or working on my research.

Last but not least, I would hereby like to thank my parents and sisters for their support and patience. Thanks to Cathelijne, my oldest sister, many spelling mistakes have been corrected. Finally, I would like to thank Anne, my girl friend, for coping with me while I was grumpy, working late, programming all the time, and writing my thesis.

Sebastiaan Tesink
Tilburg, February 28 2007

Contents

Chapter 1

Introduction

The threat of computer viruses costed an estimated 15 billion Euro in 2004 in the United Kingdom. A wider connectivity among companies increases the exposure to this threat every year. In order to prevent viruses and other security threats infecting computers, several precautions can be taken. However, "businesses are still finding their precautions are inadequate" (BBC News, 2004).

One possible precaution is the use of an Intrusion Detection System (IDS). The main objective of an Intrusion Detection System is to detect all intrusions, and only intrusions, in an efficient way (Gowadia et al., 2005). This may lead to an earlier detection of viruses and worms, and an early warning system in case of a computer virus outbreak. An effective IDS has to distinguish between incidents and "normal" alerts (Huang et al., 1999). Furthermore, it is important for an IDS to be efficient, i.e. that the number of false positives and false negatives are reduced (Gowadia et al., 2005). This means that while the number of false alarms should be reduced, real attacks should not go unnoticed.

There are many similarities between an IDS and a spam filter. Ideally, a spam filter labels spam e-mails as spam, leaving the legitimate e-mails unlabelled. A user may provide feedback to the spam filter by either correcting the spam filter, or by agreeing with the spam filter. This feedback is used by the spam filter to decrease the number of false positives and false negatives. An IDS, on the other side, labels alerts as incidents or as non-incidents. In an ideal situation, the user may provide feedback by (dis)agreeing with the IDS.

The input of an IDS is provided by one or more sensors, which are observation points on the network (Rietta, 2006). These sensors normally generate a lot of alerts (Varine, 2001). Not all of these alerts are relevant however. All alerts are analysed, and only the relevant alerts are reported as incidents. The overview of this process is depicted in Figure 1.1. The input for this process, consisting of alerts, is provided by the sensors.

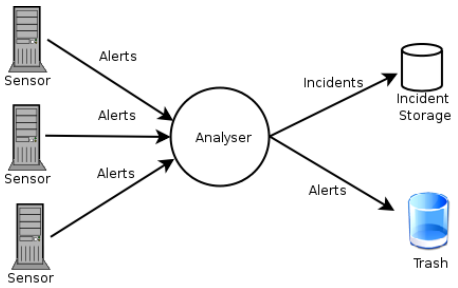

Figure 1.1: *Sensors provide alerts that are divided into alerts and incidents by the analyser.*

For the analysis process described in Figure 1.1, machine learning techniques may be a useful application. There are several reasons for this. First of all, machine learning techniques can be effective in environments in which the task cannot easily be defined properly except by example

(Nilsson, 1996). Furthermore, it is hard, if not impossible for human beings to find relationships and correlations in vast amounts of data (Nilsson, 1996). During this research, an attempt is being made to filter incidents from alerts. This classification will be carried out by various machine learning techniques.

1.1 Objective and Research Question

The objective of this research is "to find a machine learning technique that helps to distinguish between alerts and incidents based on logging information taken from Intrusion Detection Systems, in order to improve incident detection".

The research question that will be answered during the research will be **"Is it possible to distinguish computer security alerts from computer security incidents by using Intrusion Detection System logging and machine learning techniques, and what would be an appropriate technique?".**

The subquestions that can be derived from this research question are:

1. What is an Intrusion Detection System?
2. What is a computer security alert, what is a computer security incident and what are the differences between them?
3. What kind of different types of machine learning techniques exist?
4. Which technique fits best in the situation of incident determination?
5. Is machine learning an effective tool to determine computer security incidents based on IDS input?
6. Does machine learning offer genuine knowledge discovery that is intelligible for human beings?

An effort will be made to answer these subquestions throughout this thesis.

1.2 Scope

The labels of the training and test data for the experiments described here, are made available by UvT-CERT[1], which is the Computer Security Incident Response Team (CSIRT) of Tilburg University. A CSIRT is "a service organisation that is responsible for receiving, reviewing, and responding to computer security incident reports and activity" (West-Brown et al., 2003). The terms CERT and CSIRT will be used interchangeably throughout this thesis. By providing this information, the people of UvT-CERT indirectly trained the machine learning classifiers, as will be described later. For other CERT-teams, this approach may be useful as well. This way CERT-teams may train their own personalised machine learner. This will be discussed more elaborately throughout this thesis.

Future work may be integrated into AIRT[2], which is an abbreviation for Application for Incident Response Teams, which is actively developed at the InfoLab of Tilburg University, in cooperation with SURFnet[3]. AIRT is developed for security teams with a small or medium volume of security incidents, which means about 100 security incidents a day, which is typical for medium size universities and organisations (Tesink, 2005). There are some security teams that have a much larger volume of incidents, which may lead to the solutions presented here possibly being ineffective for such teams.

It may be useful to integrate the outcome of the experiments with AIRT. That way, alerts may be automatically classified into incidents, which saves the CERT-team a lot of time. Furthermore, people of the CERT-team may train their own machine learner, by giving feedback on the classification, much as with a spam filter. This feedback may be either "this alert was incorrectly

[1] http://www.tilburguniversity.nl/services/its/security/uvt-cert/
[2] http://www.airt.nl
[3] http://www.surfnet.nl

classified as an incident", or "although not classified as an incident, it was incident". This way, the machine learner can learn from its own mistakes incrementally.

The experiments conducted here are used as a proof of concept however. In case these experiments are proven to be successful, a subsequent evaluation may be made whether or not to integrate it into AIRT.

1.3 Methodology

By no means, this thesis is meant to describe the complete field of both IDSs and machine learning in full depth. Instead, it will focus on the parts that were considered to be of value for the research, and that were necessary to answer the research questions.

1.3.1 Literature

First of all, the existing literature in the field of IDS will be cited as a basis for answering the main research question. Although there is a particularly low number of scientific articles, an effort will be made to select relevant articles.

Next to the literature on IDS, literature on machine learning techniques will be used as well. This literature will be used to determine the appropriate techniques for processing the IDS output. Within the literature on machine learning techniques, several ideas are mentioned with respect to this subject. Therefore, many of these ideas will be evaluated and discussed.

1.3.2 Implementation

In order to evaluate different machine learning techniques and their working with IDS input, three different strategies for machine learning will be implemented. The first one will be a method based on eager learning. The second technique will be based on lazy learning. After having implemented both machine learning strategies, a hybrid model, which is a combination of both models, will be implemented and evaluated as well. These different learning strategies are expected to have different outcomes, as will be explained later. These different outcomes will be evaluated and compared to the available literature.

For the implementation, historical data of the classification of incidents was needed. This offers an insight in how security professionals classified alerts by hand. The CSIRT of Tilburg University used the input of several sensors, to label the alerts either as normal behaviour or as an incident. This labelled information will be used to train and test the machine learning algorithm. The training set that will be used will contain log information of more than one year. The exact amount of log information, as well as the exact implementation of the three different strategies mentioned above, will be determined during the experiment.

1.3.3 Validation

In order to be efficient, it is important for an IDS that both the number of false positives and the number of false negatives are reduced (Gowadia et al., 2005). For this reason, important parameters to evaluate are the accuracy, precision, recall and F-measure. These metrics will be discussed in Chapter 4.

The evaluation of the measurements found during the experiments will be the subject of Chapter 5. The outcomes will be analysed, and compared to results that were found in the literature.

1.4 Outline

The chapters of this thesis reflect the different aspects of the research cycle. In order to gain a better insight in the field of Intrusion Detection Systems, Chapter 2 will focus on the goals, functions and types of IDSs. Furthermore, the main subject of this chapter will be the definition of alerts, incidents and information collected by an IDS. Based on the collected information a classification will be made between alerts and incidents.

Chapter 3 will outline different machine learning techniques that could be applied for this classification experiment. Two different approaches will be discussed; lazy and eager learning. Furthermore, it will focus on hybrid models, which combines both lazy and eager learning methods.

The topic of Chapter 4 is the research design used for the experiments. The features and classes used for the classification process will be discussed during this chapter. Furthermore, the training and data sets, as well as the research method will be point of discussion. Finally, the classification methods and the tools that will be used during the research form the final part of Chapter 4.

An overview of the results of the research will be given in Chapter 5. This overview will be accompanied with an analysis of the measurements.

In Chapter 6, general conclusions will be drawn, based on the previously mentioned chapters, and references to future work will be made.

Chapter 2

Intrusion Detection Systems

This chapter aims to give a theoretical framework and introduction to intrusion detection systems. First of all, a description of IDS will be given, followed by a discussion of its components. After this first introduction, the focus will shift to the definitions of alerts and incidents, used throughout this thesis. The functions, goals and different types of IDS will be the final issues of this chapter.

2.1 Description of IDS

The growth of network intrusions on large enterprise networks continues to increase. Thousands of hackers probe and attack computer networks each day. These attacks range from relatively benign ping sweeps to sophisticated techniques exploiting security vulnerabilities (Jackson et al., 2004).

Intrusion detection is the task of detecting and responding to this kind of computer misuse, by detecting unauthorised access to a computer network (Proctor, 2001). Intrusion detection systems are "systems that collect information from a variety of system and network sources, and then analyse the information for signs of intrusion and misuse" (Proctor, 2001).

In other words, an IDS is a device, typically a designated computer system, that monitors activity to identify malicious or suspicious alerts. An IDS can be compared with a spam filter, that raises an alarm if specific things occur (Pfleeger and Pfleeger, 2003).

2.1.1 IDS components

The Common Intrusion Detection Framework (CIDF) is a commonly adopted framework that describes the working of an Intrusion Detection System. The CIDF splits an IDS into four components, as shown in Figure 2.1, which are (Staniford-Chen et al., 1998):

- Sensors
- Analysers
- Incident storage
- Response units

These basic components of an IDS will be described in more detail below.

Sensors

The role of the sensors is to obtain alerts from the larger computational environment outside the intrusion detection system, and provide them in the CIDF to the rest of the system (Staniford-Chen et al., 1998). These event generators are often referred to as event generators.

Most modern intrusion detection systems employ multiple intrusion sensors to maximise their trustworthiness. It is important to fuse the different outputs of these sensors in an effective and

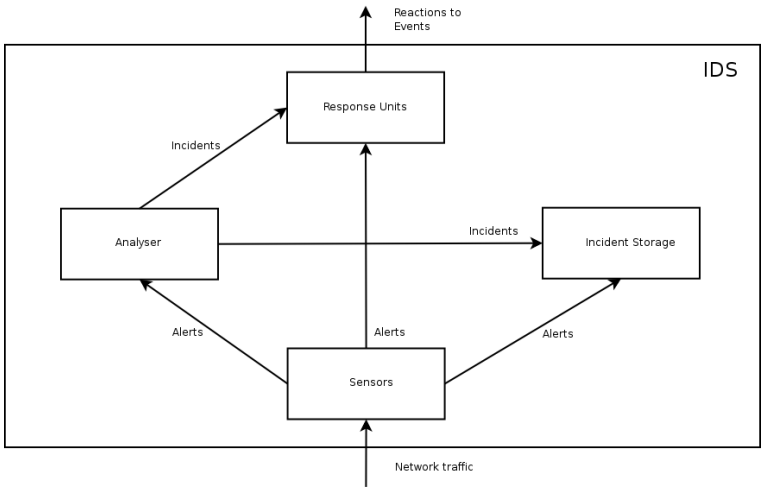

Figure 2.1: *Common components of the Common Intrusion Detection Framework (adapted from (Staniford-Chen et al., 1998))*

intelligent manner in order to provide the security administrator with an "overall security view" that can serve as an aide to appraise the trustworthiness in the multi-sensor IDS (Siraj et al., 2004). Generally speaking, there are two distinct types of sensors; network-based sensors and host-based sensors. These types of sensors will be the topic of Section 2.4.1.

Analysers

Analysers use the input of the sensors, analyse the information gathered by these sensors, and return a synthesis or summary of the input (Staniford-Chen et al., 1998). Today, artificial intelligence has become an indispensable tool in the analysers of intrusion detection systems (Siraj et al., 2004). There are two types of analysers; pattern-based analysers and anomaly-based analysers. Both types of analysers will be discussed in Section 2.4.

Incident storage

The incident storage stores all alerts, supporting the analysis process. During the analysis process instances of alerts stored in the incident storage can be used in order to classify an alert. Databases are not expected to change or process the alerts in any way (Staniford-Chen et al., 1998).

Response units

Response units (either teams of humans or automated processes) carry out prescriptions controlled by the analyser, that instructs them to act (Staniford-Chen et al., 1998). The response unit may for example send an e-mail to a system-administrator to inform him/her of suspicious alerts. Another possible response could be to adapt firewall rules in order to prevent further intrusions.

Now that the different components of an IDS are discussed, an overview of its goals and functions will be given.

2.2 Goals and functions of an IDS

Many studies have shown that most computer security incidents are caused by insiders, i.e. people who would not be blocked by a firewall. Since insiders require access with significant privileges to do their daily jobs, this results in the need for extra security measures within the organisation.

IDSs may complement other preventive controls (e.g. firewalls) as the next line of defence within the organisation (Pfleeger and Pfleeger, 2003). An IDS is a device that is placed inside a protected network to monitor what occurs within the network. This is schematically shown in Figure 2.2. An IDS offers the opportunity to detect an attacker that is able able to pass through the router and pass through the firewall. The detection can take place at the beginning of the attack, during the attack, or after it has occurred. IDSs activate an alarm, which can take defensive action (Jackson et al., 2004).

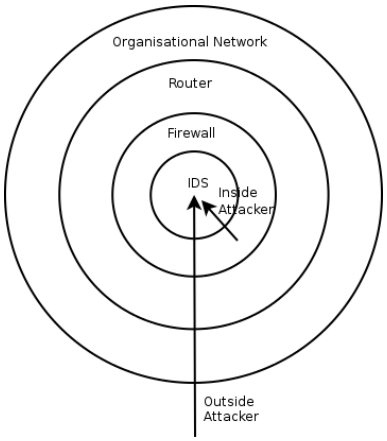

Figure 2.2: *Illustration of the placement of an IDS within an organisational network (based on (Jackson et al., 2004))*

The goal of intrusion detection systems is to accurately detect anomalous network behaviour or misuse of resources (i.e. incidents), sort out true attacks from false alarms, and notify network administrators of the activity. Many organisations now use intrusion detection systems to help them determine if their systems have been compromised[1].

Given the goal of an IDS, the functions of an IDS can be (Pfleeger and Pfleeger, 2003):

- Monitoring users and system activity
- Auditing system configuration for vulnerabilities and misconfigurations
- Assessing the integrity of critical system and data files
- Recognising known attack patterns in system activity
- Identifying abnormal activity through statistical analysis
- Managing audit trails and highlighting user violation of policy or normal activity
- Correcting system configuration errors
- Installing and operating traps to record information about intruders

An IDS may embody one or more of these functions, depending on the type of IDS. The above-mentioned functions help support the organisation's security teams, such as Computer Emergency Response Teams (CERTs), also known as Computer Security Incident Response Team (CSIRT).

2.3 Alerts and Incidents

In this thesis, the terms "alerts" and "incidents" play an important role. The distinction between alerts and incidents will be clarified in this section.

2.3.1 Alerts

Alerts (sometimes referred to as indicators or events) are defined as observable or discernible actions that confirm or deny enemy intentions (Bejtlich, 2004). With regard to the framework presented

[1] `http://www.sei.cmu.edu/news-at-sei/features/2001/1q01/pdf/feature-3-1q01.pdf`

in Figure 2.1, the alerts are the input of the event generators.

Throughout this thesis, alerts will be seen as all the traffic that was picked up by the IDS sensors. A normal IDS has no special features. It is just a computer system or a network segment, with no special production systems running on it (Pfleeger and Pfleeger, 2003). Since an IDS does not offer any useful services to Internet users and the Internet addresses of the IDS are not publicly known, there should theoretically be no traffic to or from the IDS and therefore most traffic on the IDS is suspicious (Krasser et al., 2005; Jackson et al., 2004).

2.3.2 Incidents

All alerts have a certain risk value, but some have greater value than others (Bejtlich, 2004). The alert that has a high enough value, and which is considered to be a "security-relevant system event in which the system's security policy is disobeyed or otherwise breached" (Shirey, 2000) is considered to be an incident. There are some problems however in distinguishing incidents from alerts.

First of all, incidents are the results of an analyst's interpretation of indicators. Analysts scrutinise the alerts generated by the event generators (Proctor, 2001). It is the human brain that decides on the value of an alert. Identification of an incident is a manual process where specialised security analysts must observe and analyse unusual activity (Berk et al., 2000). Intrusion Detection software builds patterns of normal system usage, triggering an alarm any time the usage seems abnormal (Jackson et al., 2004).

Secondly, as the amount of traffic on a network grows, monitoring that traffic and isolating threats from normal or otherwise harmless activity becomes increasingly difficult. The situation is even worse for large universities where the need for academic freedom prevents network administrators from implementing strict controls on their networks (Jackson et al., 2004).

Because of the above-mentioned difficulties, IDSs may return the wrong result. The following two types of false results may occur (Pfleeger and Pfleeger, 2003):

Type I	False positive	An alarm is raised for something that is not really an attack.
Type II	False negative	Not raising an alarm for a real attack.

Table 2.1: *Types of errors that may occur, using an IDS (Pfleeger and Pfleeger, 2003)*

The degrees of false positives and false negatives together represent the sensitivity of the system. Most IDS implementations allow to tune the system's sensitivity, towards one or the other. These degrees of false positives and false negatives are closely related to the terms recall and precision respectively. A higher degree of false positives means a lower precision, and vice versa. A higher degree of false negatives means that the recall of the IDS is lower. The terms precision and recall will be looked at more closely in Section 4.6.

2.4 Types of IDS

Both for the data collection (or event generators) as well as the analysis of the alerts, there are two approaches. For the collection of data, there is a network-based IDS approach, and a host-based IDS approach. For alert analysis, IDSs may be either signature-based or heuristic. In order to gain a better insight into current IDSs, all these different methods will be mentioned briefly. After having discussed the above-mentioned approaches, the focus will be on one specific type of implementation; honeypots. These honeypots will be described in more detail in Section 2.4.3.

2.4.1 Types of Data collection

As already pointed out, the data collection of IDS devices can be network-based or host-based.

Network-based IDS

A network-based IDS is a stand-alone device attached to the network to monitor traffic throughout that network (Pfleeger and Pfleeger, 2003). Intrusion detection is considered to be network-based when the system is used to collect and analyse network packets. Network packets are usually "sniffed" off the network, and are preferably derived straight from the output of routers and switches (Proctor, 2001).

Host-based IDS

A host-based IDS runs on a single workstation or client or host, to protect that specific host (Pfleeger and Pfleeger, 2003). Intrusion detection is host-based when the system is used to analyse data that originates on computers (hosts), such as application and operating system event logs. This is in contrast to network-based intrusion detection, is used to process data that originates on the network. Host-based intrusion detection is particularly effective at detecting insider misuse because of the target data source's proximity to the authenticated user (Proctor, 2001).

2.4.2 Types of Analysis

In general, there are two types of Intrusion Detection Systems; pattern-based and anomaly-based systems (Stillerman et al., 1999). The difference between these two systems lies in the type of detection used during the analysis phase mentioned in Figure 2.1.

Pattern-based Systems

Pattern-based, often referred to as signature-based, IDS perform simple pattern matching and report situations that match a pattern corresponding to a known attack type. Anomaly-based IDSs, also known as heuristic IDSs, build a model of acceptable behaviour and flag (i.e. label) exceptions to that model; for the future, the administrator can mark a flagged behaviour as acceptable so that the anomaly-based IDS will now treat that previously unclassified behaviour as acceptable (Pfleeger and Pfleeger, 2003). Most commercial IDSs utilise this approach (Botha and von Solms, 2003).

The main advantage of pattern-based systems is that it has a relatively low rate of false alarms (Stillerman et al., 1999), which means that the IDS has a relatively high precision. This high precision is caused by the fact that a pattern based system is explicitly programmed to detect certain known kinds of attacks (Stillerman et al., 1999).

The main disadvantage of this kind of systems, however, is that the detection rate of attacks is relatively low (Stillerman et al., 1999). Derived from that, one could assume a lower recall for new types of intrusions. This is the result of the following two problems of pattern-based systems.

The first problem with signature-based detection are the signatures themselves. An attacker will try to modify a basic attack in such a way that it will not match the known signatures of that attack. The attacker may insert malformed packets that the IDS will see, to intentionally cause a pattern mismatch; the protocol handler stack will then discard the packets because of the malformation. Each of these variations could be detected by an IDS, but more different signatures require additional work for the IDS, which reduces performance (Pfleeger and Pfleeger, 2003).

Secondly, signature-based IDSs cannot detect a new attack for which a signature is not yet installed in the database. Ideally, signatures should match every instance of an attack, match subtle variations of the attack, but not match traffic that is not part of an attack. However, this goal is difficult to accomplish in current IDSs (Pfleeger and Pfleeger, 2003).

Anomaly-based Systems

Because signatures are limited to specific, known attack patterns, so-called heuristic intrusion detection becomes useful. Instead of looking for matches, heuristic intrusion detection looks for

behaviour that is suspicious (Pfleeger and Pfleeger, 2003). Anomaly-based systems attempt to characterise normal operation, and try to detect any deviation from normal behaviour (Stillerman et al., 1999). This form of intrusion detection is also known as misuse intrusion detection. The principle of this sort of intrusion detection is that the real activity is compared against a known suspicious area of alerts (Pfleeger and Pfleeger, 2003).

The advantage of an anomaly-based system is a relatively high detection rate for new types of intrusions, i.e. a higher recall. The main disadvantage of anomaly-based intrusion detection, on the other hand, is a higher rate of false alarms as well, which means a lower precision (Stillerman et al., 1999). Another disadvantage is that this type of IDSs tends to be computationally expensive because several metrics are often maintained that need to be updated against every system activity and, due to insufficient data, they may gradually be trained incorrectly to recognise an intrusive behaviour as normal due to insufficient data (Botha and von Solms, 2003).

2.4.3 Honeypots

During this research, the focus will be on network-based IDS; host-based IDS are considered to be out of scope. Moreover, the classification process of alerts will be the main focus of the research. The input for the analysis will be provided by honeypots, which can be considered to be the event generators of the research. Therefore these honeypots will be briefly discussed.

Honeypots are collections of networked computer systems which are intended to be attacked and broken into in an observed fashion, keeping track of any (mis-)use (May and Dornseif, 2004). These honeypots are closely monitored network decoys serving several purposes (The Honeynet Project, 2002):

- they can distract adversaries from more valuable machines on a network;
- they can provide early warnings about new attack and exploitation trends;
- they allow in-depth examination of adversaries during and after exploitation of a honeypot.

An example of a honeypot is a system used to simulate one or more network services that you designate on your computer's ports. An attacker assumes you're running vulnerable services that can be used to break into the machine. This kind of honeypot can be used to log access attempts to those ports including the attacker's keystrokes (The Honeynet Project, 2002).

As discussed in Section 2.3.2, to distinguish incidents from alerts one needs a specialised security professional. Moreover, as the amount of network traffic increases, such as in university networks, it is becoming increasingly difficult to make a clear classification. In Chapter 1, however, it was mentioned that machine learning techniques can be effective in environments in which the task can only be defined by many examples. Secondly, machine learning may help to find relationships and correlations in vast amounts of data that a human being would simply overlook. Because of its potential usefulness for the classification of alerts, the next chapter will discuss machine learning.

Chapter 3

Machine Learning

As the use of machine learning for the classification of alerts may not be evident at first sight, this chapter will discuss what machine learning is, and why it is used for this specific classification task. After this discussion, the differences between supervised and unsupervised learning will be given, followed by two types of machine learning; eager and lazy learning. For both eager and lazy learning, different algorithms and their characteristics will be mentioned. The description of hybrid machine learning will be the final topic of this chapter.

3.1 What is Machine Learning

When a computer needs to perform a certain task, a programmer's solution is to write a computer program that performs the task. A computer program is a piece of code that instructs the computer which actions to take in order to perform the task.

The field of machine learning is concerned with the higher-level question of how to construct computer programs that *automatically learn with experience* (Mitchell, 1997a). A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E (Mitchell, 1997a). Thus, machine learning algorithms automatically extract knowledge from machine readable information (Hall and Smith, 1996). In machine learning, computer algorithms (learners) attempt to automatically distill knowledge from example data. This knowledge can be used to make predictions about novel data in the future and to provide insight into the nature of the target concepts (Hall and Smith, 1996).

Applied to the research at hand, this means that a computer would learn to classify alerts into incidents and non-incidents (task T). A possible performance measure (P) for this task would be the accuracy with which the machine learning program classifies the instances correctly. The training experiences (E) could be labelled instances. All of these will be elaborated more thoroughly in Chapter 4.

3.2 Benefits of Machine Learning

In particular, machine learning plays an essential role in the following three areas of software engineering (Mitchell, 1997a):

1. Data mining problems where large databases may contain valuable implicit regularities that can be discovered automatically.
2. Difficult-to-program applications, which are too difficult for traditional manual programming.
3. Software applications that customise to the individual user's preferences, such as personalised advertising.

There are several reasons why machine-learning plays a role in these three domains. First of all, for the classification of security incidents, a vast amount of data has to be analysed containing historical data, as was mentioned in Section 2.3.2. It is difficult for human beings to find a pattern in such an enormous amount of data. Machine-learning, however, seems well-suited to overcome this problem and can therefore be used to discover those patterns.

With respect to the difficult-to-program applications, an analyst's knowledge is often implicit, and the environments are dynamic (Pietraszek, 2004). As a consequence, it is very hard to program an IDS using ordinary programming languages that require the explicitation and formalisation of knowledge. The adaptive and dynamic nature of machine-learning makes it a suitable solution for this situation.

Third, the environment of an IDS and its classification task highly depend on personal preferences. What may seem to be an incident in one environment may be normal in other environments (Mitchell, 1997a). This way, the ability of computers to learn enables them to know someone's "personal" (or organisational) preferences, and improve the performance of the IDS, for this particular environment.

3.3 Supervised versus Unsupervised Learning

Machine learning can be divided in two categories; supervised and unsupervised machine learning algorithms (Hendrickx, 2005). In supervised learning, the input of the learning algorithm consists of examples (in the form of feature vectors) with a label assigned to them. The objective of supervised learning is to learn to assign correct labels to new unseen examples of the same task.

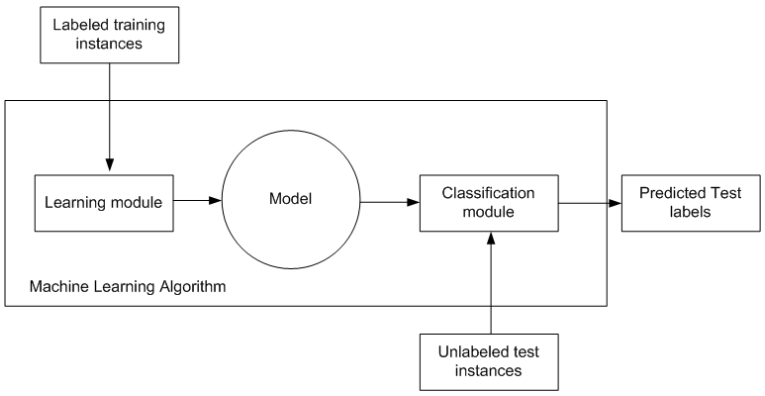

Figure 3.1: *Supervised machine learning, based on (Hendrickx, 2005)*

As shown in Figure 3.1, a supervised machine learning algorithm consists of three parts: a learning module, a model and a classification module. The learning module constructs a model based on a labelled training set. This model consists of a function that is built by the learning module, and contains a set of associative mappings (e.g. rules). These mappings, when applied to an unlabelled test instance, predict labels of the test set. The prediction of the labels of the test set is done by using the classification module.

In contrast to supervised learning, in unsupervised learning the machine simply receives inputs, but obtains neither supervised target outputs, nor rewards from its environments. Unsupervised algorithms learn from unlabelled examples. Unsupervised learning can be thought of as finding patterns in the data and beyond what would be considered pure unstructured noise (Ghahramani, 2004). The objective of unsupervised learning may be to cluster examples together on the basis of their similarity (Hendrickx, 2005).

Throughout this thesis, supervised learning methods will be used. As will be described in Section 4.1 the input of the classification model used in this research was described by labelled information.

3.4 Eager versus Lazy Learning

Another distinction between types of machine learning is the one between eager versus lazy learning. As will be explained below, both types of learning have different characteristics.

3.4.1 Eager learning

Eager learning is a form of supervised learning, which means that there is a learning module, a model and a classification module, as shown in Figure 3.1. Eager learning algorithms invest most of their effort in the learning phase. They construct a compact representation of the target function by generalising from the training instances. Classification of new instances is usually a straightforward application of simple learned classification rules that employ the eager learner's model (Hendrickx, 2005).

A method is called eager when it generalises beyond the training data before observing a new query, committing at training time to the network structure and weights that (i.e. the model) define its approximation to the target function (Mitchell, 1997b).

Rule induction

Rule induction is a form of eager learning. During the learning phase, rules are induced from the training sample, based on the features and class labels of the training samples. The goal of rule induction is generally to induce a set of rules from data that captures all generalisable knowledge within that data, and that is as small as possible at the same time (van den Bosch, 2004). The rules that are extracted during the learning phase, can easily be applied during the classification phase when new unseen test data is classified.

There are several advantages of rule induction. First of all, the rules that are extracted from the training sample are easy to understand for human beings. The rules are simple if-then rules. Secondly, rule learning systems outperform decision tree learners on many problems (Cohen, 1995).

One disadvantage of rule induction, however, is that it scales relatively poorly with the sample size, particularly on noisy data. This means that, providing a training set B to a rule induction classifier, which is twice as large as training set A, the computational time it takes to induce rules is more than twice as much for training set B. Given the prevalence of large noisy datasets in real-world applications, this problem is of critical importance (Cohen, 1995).

3.4.2 Lazy learning

Next to eager learning, there is also lazy learning as a form/variant of supervised learning. In different contexts, memory-based learning algorithms have been named lazy, instance-based, exemplar-based, memory-based, case-based learning or reasoning (van den Bosch and Daelemans, 1998). The reason for calling certain machine learning methods lazy, is because they defer the decision of how to generalise beyond the training data until each new query instance is encountered (Mitchell, 1997b).

A key feature of lazy learning is that during the learning phase, all examples are stored in memory and no attempt is made to simplify the model by eliminating noise, low frequency events, or exceptions (Daelemans et al., 1999). The learning phase of a lazy learning algorithm consists simply of storing all encountered instances from a training set in memory (van den Bosch and Daelemans, 1998). The search for the optimal hypothesis takes place during the classification phase (Hendrickx, 2005).

On being presented with a new instance during the classification phase, a memory-based learning algorithm searches for a best-matching instance, or, more generically, a set of the k best-matching instances in memory. Having found such a set of k best-matching instances, the algorithm takes the (majority) class and the instances in the set are then labelled as belonging to the class of the

new instance. Pure memory-based learning algorithms implement the classic k-nearest neighbour algorithm (van den Bosch and Daelemans, 1998), that will be explained in more detail below.

A variant of memory-based learning/lazy learning is similarity-based learning. Similarity-based learners represent knowledge in the form of specific cases or experiences. They rely on efficient matching methods to retrieve these stored cases so they can be applied in novel situations. Similarity-based learners are usually computationally simple, and variations are often considered as models of human learning (Hall and Smith, 1996).

Instance-based learning generates classification predictions using only specific instances, rather than pre-compiled abstractions during predictions tasks. Instance-based learning algorithms do not maintain a set of abstractions derived from specific instances. This approach extends the nearest neighbour algorithm which has large storage requirements (Aha et al., 1991).

It appears that in order to learn tasks successfully, a learning algorithm should not forget any information contained in the learning material and it should not abstract from the individual instances. Forgetting instance tokens and replacing them by instance types may lead to considerable computational optimisations of memory-based learning, since the memory that needs to be searched may become considerably smaller (van den Bosch and Daelemans, 1998). One disadvantage of lazy learning, however, is that noise in the training data can harm accurate generalisation (Daelemans et al., 1999).

The reason for applying lazy learning algorithms for this research despite its disadvantages, is that they have performed extremely well in domain-specific systems that were used in industrial applications (Aha et al., 1991). An explanation for this is that using specific instances in supervised learning algorithms decreases the costs incurred when updating concept descriptions, increases learning rates, allows for the representation of probabilistic concept descriptions, and focuses theory-based reasoning in real-world applications (Aha et al., 1991).

k-Nearest Neighbour

There are several instance-based learning algorithms. One of the best known is k-Nearest Neighbour (k-NN), which will be used here. Historically, memory-based learning algorithms are even descendants of the k-nearest neighbour (henceforth k-NN) algorithm (Daelemans et al., 2005).

The learning phase of k-NN is simply a storage step. As was described earlier, the most important phase for a lazy learner is the classification phase. The k-NN algorithm uses all labelled training instances as a model of the target function. During the classification phase, k-NN uses a similarity-based search strategy to determine a locally optimal hypothesis function. Test instances are compared to the stored instances and are assigned the same class label as the k most similar stored instances (Hendrickx, 2005).

The similarity or distance between two instances A and B is measured according to the distance function:

$$\Delta(A, B) = \sum_{i=1}^{n} w_i \delta(a_i, b_i).$$

In this equation, n is the number of features, w_i is a global weight for feature i, and distance metric δ estimates the difference between the feature values of the two instances. Distance metric δ and feature weighting w are both algorithmic parameters besides k (Hendrickx, 2005).

3.4.3 Differences between Eager and Lazy learning

Concluding from the previous sections, there are three distinctions between eager and lazy learning. First of all, lazy learners defer processing of their inputs until they receive requests for information; they simply store their inputs for future use (Aha, 1997). This means that eager learners put effort in the learning phase, whereas lazy learners divert this effort to the classification phase (Hendrickx, 2005).

Secondly, lazy learning algorithms reply to information requests by combining their stored data (Aha, 1997). In other words, eager learners form a global hypothesis (or model, as shown in

Figure 3.1) or the target function that describes the complete instance base. Lazy learners, however, produce for each new instance a local hypothesis, based on a part of the instance base (Hendrickx, 2005).

Finally, lazy learners discard the constructed answer and any intermediate results (Aha, 1997). The hypothesis constructed by an eager learner is independent of the new instance to be classified. The hypothesis of the lazy learner, however, depends on the new instance (Hendrickx, 2005).

This leads to the fact that on the one hand, lazy algorithms have lower computational costs than eager algorithms during training. On the other hand, they typically have greater storage requirements and often have higher computational costs when answering requests (Aha, 1997).

3.5 Hybrids

Next to lazy learning algorithms and eager learning algorithms, hybrids of both types were evaluated as well. The hybrids are mixtures of the k-NN classifier and rule induction. The reason for constructing hybrids is the contrast between memory-based learning and eager learning. Memory-based learners put time in the classification phase, whereas eager learners invest their time in the learning phase. Combining eager and lazy learners into hybrids, will produce machine learners that put effort in both the learning phase and the classification phase. This leads to the expectation that this double effort will be repaid with improved performance. The hybrid will use both the global hypothesis as induced by rule induction, as well as the local hypothesis created during memory-based learning (Hendrickx, 2005).

During this research, k-NN will be combined with rule-induction. Although rules appear quite different object from instances as used in k-nearest neighbour classification or instance-based learning, there is a continuum between them. Rules can be seen as generalised instances; they represent a subset of training instances that match on the conditions on the left-hand side of the rule. Therefore, k-NN classification can naturally be applied to rules (van den Bosch, 2004).

An extension to the k-nearest neighbour classifier is described in which automatically induced rules are used as binary features, which are active in an instance when the left-hand side of the corresponding rule matches with the instance. The RIPPER rule induction algorithm is employed to produce the rules. The similarity between a memory instance and a new instance is based on the rules the two instances share. In other research (e.g. (van den Bosch, 2004; Hendrickx, 2005)), some significant improvements in k-NN classification are observed, particularly with artificial benchmark tasks (van den Bosch, 2004).

3.5.1 Rules-R-H

In order to create a hybrid, based on Rules-R-H, rules are induced from the training set using RIPPER. These rules are then transformed into vectors, representing the binary rule-features. The vectors that are produced can be used as instances of a training set. The test set is converted to a vector of binary rule-features as well, and classified using k-NN classification (van den Bosch, 2004).

F1	F2	F3	F4	Class
t	f	t	t	y
f	t	f	t	n
f	f	f	t	n
t	t	f	f	y

Table 3.1: *Values of features F1, F2, F3, F4 and class value of four example instances*

This conversion, or mapping, works as follows. The rule set is represented as a vector of bits (binary rule-features) where each each bit represent a certain rule (Hendrickx, 2005). Thus, the number of features is equal to the number of rules induced by RIPPER. This conversion is shown

with the example instances of Table 3.1. The example shows four instances, with four features, and the classes of the instances. When RIPPER was applied to the entire set, the following RIPPER hypotheses were produced:

```
y IF F3 = t F4 = t .
y IF F1 = t F4 = f .
y IF F2 = f F4 = t .
n IF .
```

Now applying the first rule on the first instance, the first rule fires, since the third feature is "t", and the fourth feature is "t". This results in 1 for the binary rule-features. Applying the second rule on the first instance, doesn't result in a "firing" IF-statement, since feature 1 is not "t", and a 0 is filled in for its recoded feature value. Looping through the rules for each instance, will result in the binary rule-features of Table 3.2.

An instance base that has been transformed in the way described above can be used again as training material for a k-NN classifier. Like the k-NN classifier trained on the instance base before the transformation, this new k-NN classifier can compute distances between transformed instances with any distance (van den Bosch, 2004).

F1'	**F2'**	**F3'**
1	0	1
0	0	0
0	0	1
0	1	0

Table 3.2: *Recoded feature values*

When using Rules-R-H, the binary rule-features replace the original features in the instances. This hybrid is referred to as Rules-R-H, because the middle R denotes "replace" (Hendrickx, 2005).

From the k-NN perspective, Rules-R-H attempts to repair k-NNs sensitivity to noise and irrelevant features, since induced rules will typically not cover low-frequency noise and will not test on irrelevant features (Hendrickx, 2005). Replacing the original features of the instances by rule-features can thus be considered as a compression and noise filtering step in which the rule induction algorithm has grouped interacting feature values together, which the normal k-NN algorithm is incapable of (Hendrickx, 2005).

3.5.2 Rules-A-H

Next to Rules-R-H, there is a second type of hybrid. When using this hybrid, the initial features of the instance are not replaced by the binary rule-features. These binary rule-features are added to these initial features. This is also the reason why this hybrid is called Rules-**A**-H. Thus, this hybrid is a k-NN classifier with extra added features that represent the per-instance firing patterns of the induced rule set, the same way as described for Rules-R-H. In this case the rule-features can not be considered as a compression and noise filtering step, but it is expected that the rule-features can repair k-NNs sensitivity to noise in a more implicit way (Hendrickx, 2005).

As was described in Section 3.4.2, feature weighting in k-NN gives a higher weight to more important features. As many of the created rule-features will have a strong predictive power, they are likely to receive high feature weights, enabling them to influence the distance calculation. It is expected that the extra rule-features can overrule the influence of noise and irrelevant features (Hendrickx, 2005).

3.6 Conclusions

Now that the different types of machine learning have been discussed, the practical implementation of these machine learning techniques for IDS purposes will be the subject of Chapter 4. Having

elaborated lazy learners, eager learners, and hybrids, the hypothesis exists that combining the efforts in the learning task of eager learners with the efforts of the lazy learners' classification task will increase the accuracy with which incidents are predicted.

One of the weaknesses of memory-based learning, is its intolerance of attribute noise, and its intolerance of irrelevant attributes (Aha et al., 1991). Combining memory-based learning with eager learning into a hybrid may improve the generalisation performance of the classifier. On the other hand, one of the draw-backs of eager learning is its insensitivity, by its generalisation (Hendrickx, 2005). By combining the learning module of an eager learner with the classification module, the results of the classification task of incidents should improve using hybrid machine learning.

Chapter 4

Experiments

This chapter will focus on the experiments that were done in the light of the research described in this paper. The experiments will be used as a proof of concept. This chapter will start with a discussion of the data set that was used, and a description of the classification task. The features of the data set are elaborated, and test and training sets are defined for the classification task. In the final part of this chapter, the research method is explained.

4.1 Data set

The data set that will be used during the experiments, is taken from honeypots of Tilburg University. These honeypots are hidden in several network segments of the university network. All the honeypots log to one central storage server. Taken from this storage server are the instances used for this experiment. An example of a honeypot log is shown below. The data set contains 7.7 million alerts. Almost 60 thousand of these instances are marked as incidents, which means that the incidents belong to a minority class.

An example of a honeypot instance is the following:

```
2006-09-18-15:40:41.5704 tcp(6) S 1.2.3.4 2092 2.3.4.5 3632: 60 S [Linux 2.6 ]
```

The first part of this log format (2006-09-18-15:40:41.5704) is the timestamp the event occurred. The used protocol during this incident was TCP. The "S" is used to indicate that the packet is the start of a new connection. The ip address "1.2.3.4" is the source ip address; the ip address from which the attack is coming. The number "2092" stands for the port number of the source. The second ip address mentioned in this log ("2.3.4.5") indicates the destination ip address. This is the ip address at which the honeypot is working. The attack is on port number "3632". The packet size and flags appear after the colon in the log file, which is in this case "60 S". Comments like operating system identification via passive fingerprinting are appended to the end of the line (Provos, 2004).

4.2 Classification

The used data set was labelled by hand by security professionals of the university's security team, UvT-CERT[1]. This provides the ability to use supervised learning for this research. However, a site note should be made with regard to the labelling. The Computer Emergency Response Team consists of 7 persons, which leads to some noise in labelling, since there are no rules in the process of labelling incidents. Not only is this process of labelling instances dependent on the person that is tagging the instances, it can also be dependent on the mood and the amount of available time the person has.

[1] http://www.tilburguniversity.nl/services/its/security/uvt-cert/

This noise within the data set may lead to confusing errors, since the labelled information may be incorrect compared to the actual classification. Based on the information provided by the training information, the learning algorithms may for instance conclude that the alert is not an incident, corresponding with the label. If there is much irrelevant and redundant information present or the data is noisy and unreliable, then knowledge discovery during the training phase is more difficult (Hall and Smith, 1996).

4.3 Features

The feature set for this research consists of features that can be derived directly from the log files, and features that are indirectly derived from them. Features that can be directly derived from honeypot log files, are for example the timestamp or source ip address of the alert. Indirect features are for example the host name, that matches a source ip at a certain point in time, or the day of the week of the alert. Table A.1 of Appendix A lists all the features used for the experiment. Below, a brief discussion of the features will be given, and references to the scripts will be made. In case a value is not set for a feature during the collection phase, the value was set to "?", in order to be processed correctly.

Figure 4.1 shows how the indirect and direct features are schematically inserted into the database. These procedures will be explained in more detail below.

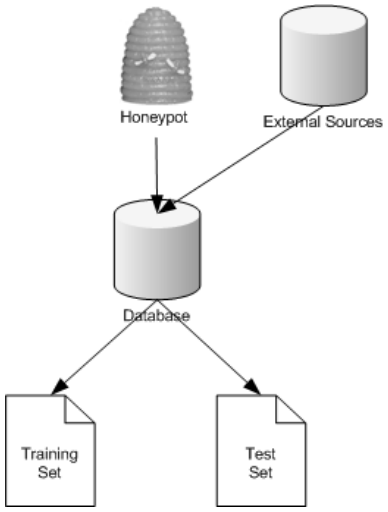

Figure 4.1: *The database contains direct input from the honeypot (direct features), as well as information provided by external sources (indirect features).*

4.3.1 Direct Features

First of all, the direct features were fed into the database, using the scripts described in Appendix B. These direct features were directly filtered from the honeypot log files. These are the timestamp, source ip address, source name, source port, destination ip address, protocol, flag, flag2, and the packet size. However, for privacy reasons, the source ip addresses were replaced by unique identifiers.

4.3.2 Indirect features

Most of the features were calculated based on the honeypot log files and external sources. Two external sources were queries, in order to gain an insight into the values of the features.

First of all, a host name lookup was done, to collect the source name, domain name and top-level domain (tld) of the alert. In the case below, the tld is "nl", the source name is "stuwww.uvt.nl", and the domain name is "uvt.nl".

```
host 137.56.0.67
67.0.56.137.in-addr.arpa domain name pointer stuwww.uvt.nl.
```

Next to this normal host name lookup procedure, a more thorough lookup was done, using the source ip address that was mentioned in the alert. The procedure of doing this is shown below.

```
# whois -h whois.cymru.com " -v 80.69.95.131 GMT"
AS    | IP           | BGP Prefix    | CC | Registry | Allocated  | Info | AS Name
20857 | 80.69.95.131 | 80.69.64.0/19 | NL | ripencc  | 2004-10-05 | GMT  | TRANSIP
```

The result of this query is the AS, BGP Prefix, CC, Registry, the time the AS number was claimed, and the AS Name of the ip address of the attacker. These features will be described below. Since the AS Name is directly derived from the AS number, the AS Name will not be used. The other features however, are added to the database, together with the previously found indirect features. This is done with the script referred to in Appendix B.

Autonomous System Numbers (ASNs) are globally unique numbers that are used to identify autonomous systems and which enable an AS to exchange exterior routing information between neighbouring ASs (American Registry for Internet Numbers (ARIN), 2006). ASs are the unit of routing policy in the modern world of exterior routing. An AS is a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy (Hawkinson, 1996). The AS could be used as an identifier for the network the attacker is from, and could therefore be a valuable feature.

The primary function of the Border Gateway Protocol (BGP) is to exchange network reachability information with other BGP systems. This network reachability information includes information on the list of Autonomous Systems (ASs) that reachability information traverses. This information is sufficient to construct a graph of AS connectivity from which routing loops may be pruned and some policy decisions at the AS level may be enforced (Rekhter, 1995). Therefore, it might be important information for the security professionals to base their classification on.

The abbreviation "CC" stands for Country Code. This is the country code of the attacker's ip address, according to the Cymru database. A security professional could use this information to classify the alert as an incident. For instance, the weight of a Dutch country code could be much higher than the weight of a Korean country code.

In general, there are five regional Internet registries Coordination Centers. These coordination centres can be RipeNCC, AfriNIC, ARIN, and LACNIC. Again, a security professional could base his decision to classify the alert as an incident on the fact whether the attacker game from an ip address under supervision of RipeNCC, for instance.

Finally, the date that the net block has been allocated or reserved is shown in the feature "allocated". This might be an indication for the person classifying alerts, about whether or not the alert is an incident.

A simple Entity Relationship Diagram (ERD) for the previously mentioned direct and derived features is shown in Figure 4.2.

4.3.3 Calculated Features

Next to the direct and indirect features, other features were calculated, using the direct and indirect features. This was done for the following direct and indirect features:

- Top level domain
- Autonomous System Numbers

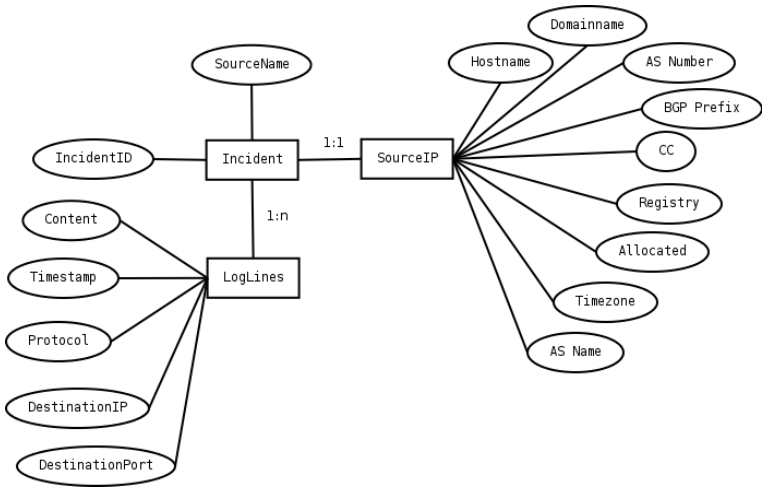

Figure 4.2: *Entity Relation Diagram of used features*

- Border Gateway Protocol
- Country Code
- Destination Port
- Domain name
- Protocol
- Source ip address
- Source port

These newly calculated features can be divided into three categories; "time differences", "boolean incident information" and "frequency information".

Intervals

For the previously mentioned features, the time between the occurrences of a certain value of a feature was measured. If for instance, the current source ip address would be 137.56.0.67, the time was calculated how long ago this value for a source ip address was seen. Not only was this done for one occurrence ago, but for the past 10 times this source ip address was observed. As mentioned before, this was also done for TLD, ASN, BGP, CC, etc.

A reference to the location where the scripts can be downloaded is shown in Appendix B.

Boolean incident information

There are two boolean features; one providing information about whether the last alert had the same feature value as this alert (e.g. same source port), and the other whether indicates whether the last time the value was seen for this feature was an incident. Both are calculated with scripts of Appendix B.

Frequency information

This type of features provides information about how often the value was seen for this feature; for instance how often the ip address "137.56.0.67" was seen. This type of features has a forward as well as a backward looking function. The frequency the value was seen for this feature is calculated backwards for the intervals:

- 5 minutes

- 1 hour
- 1 day
- 1 week
- 1 month
- 1 year
- ever

The frequency was also calculated ahead, since the incidents may be processed in batches of one day. This was done with the following intervals:

- 5 minutes
- 1 hour
- 1 day

It was anticipated that the batch would never exceed one day of instances ahead. However, since it was too computationally expensive to calculate all this frequency information for each of the 7.7 million instances, time frames of 5 minutes were created, in which all the instances were categorised.

When calculating the frequency of alerts having the AS Number "5" for the past 5 minutes, for example, the number of alerts within the same time frame can be taken. This is computationally a lot less expensive. Moreover, when calculating the number of alerts for the past hour, having AS number "5", one can add the total number of alerts in the last 12 frames of 5 minutes. This creates a sliding time window.

Next to the fact that it is faster to compute five minute frames, it also provides finer granularity. This can be illustrated by the following example. When a rule of the IDS would classify alerts as incidents when seen for 10 times per hour from the same ip address, and 9 alerts would happen at 13h59, and 9 would occur at 14h00, the IDS would not classify the alerts as incidents. In this case, the IDS tends to forget everything after one hour. With the 5 minute time frames however, the IDS has more information about what happened exactly within 60 minutes.

Time	13h00	13h05	13h10	13h15	13h20	13h25	13h30	13h35	13h40	13h45	13h50	13h55	14h00	14h05	14h10	14h15	14h20	14h25
Number of alerts	3	4	4	5	6	3	0	0	0	0	0	0	0	0	0	0	0	0
Cumulative number of alerts within 60 minutes	3	7	11	16	22	25	25	25	25	25	25	25	22	18	14	9	3	0

Table 4.1: *Example of how cumulative number of alerts during a certain period can provide finer granularity.*

An example of the use of these 5 minute frames is shown in Table 4.1. This table shows the number of alerts within every 5 minutes. Second of all, it also shows the cumulative number of alerts for the past 60 minutes. In stead of calculating the number of alerts between 13h and 14h, the cumulative number of alerts is calculated for each period of 5 minutes. This cumulative number of alerts is also shown in Figure 4.3. This figure illustrates that calculating cumulative numbers of alerts for each period of 5 minutes provides a better insight into the dispersal of alerts over the hour.

4.4 Experimental methods

In the next sections, the experimental methods will be mentioned. This section focusses on the way the training and test sets are built. The training sets will be used to train the different machine learners. The test sets will be used to evaluate the performance of the machine learners.

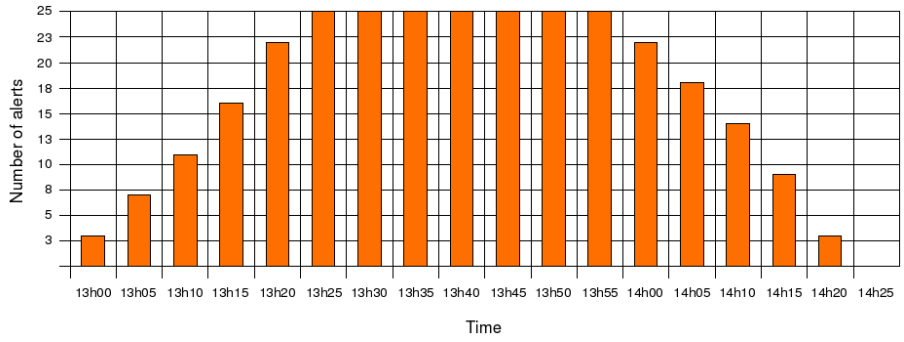

Figure 4.3: *Graph of the cumulative number of alerts within 60 minutes based on Table 4.1.*

4.4.1 Training set

Since there is a time sequence in the instances, 10-fold cross-validation cannot be used. Ten-fold cross validation would randomise the instances over time, which would destroy the time effect in the data. In stead of cross-validation, 50 points in time were determined. These points of time were taken in steps of 6 days.

Furthermore, since 7.7 million instances with 224 features each, it was virtually impossible to calculate the features of the entire data set. Therefore down-sampling was needed. Training samples are taken at random. This is done with different ratios of positive versus negative instances. All positive instances are taken in the training set, for being a minority class. The sample of negative instances is taken at random with ratios of 0.5, 1 and 2 with the positive class. This means that if there were 10.000 positive instances, 5,000, 10,000 and 20,000 negatives instances were taken respectively into the training set.

A third variable in the training set is the number of random sets. In order to be sure that the training set is randomly taken, and that (on average) this training set is not skewed, the experiment is repeated with 10 training sets.

The training and test sets are taken from the database once. For all the classifiers, the same training and test sets will be used. This is demonstrated by Figure 4.4. The scripts that were used to generate the training sets referred to in Appendix B. Training sets are generated in 10 random samples, with ratios of 0.5, 1, and 2 for 50 points in time, resulting in 3*10*50=1500 different training sets. This enables the classifiers to train over time, with different ratios of positive and negative instances.

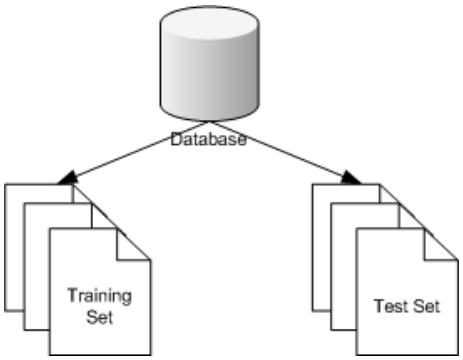

Figure 4.4: *Training and test sets are generated from the database, resulting in 1500 training sets and 500 test sets.*

4.4.2 Test set

At each point in time, a test sample is taken, containing all negative instances until a predetermined number of positive instances is encountered from this starting point. This number of positive instances that is taken in the test set is either 20, 50, 100, 200 or 500. This is done in order to estimate how well the machine learning tools can predict the outcome of the classification in time. Moreover, the fixed number of positive instances makes it possible to calculate the precision and accuracy in an unbiased way. The scripts that were used to produce the test sets are referred to in Appendix B. A total of 500 test files was produced, to evaluate the classifiers.

Figure 4.5 demonstrates the way training and test sets are picked from the instances in the database. In this case (where t=12), a sample of randomly taken negative instances, and all positive instances before the point in time where t=12 are taken. For the test set, all negative and positive instances are taken from the database, until a predetermined number of positive instances is found.

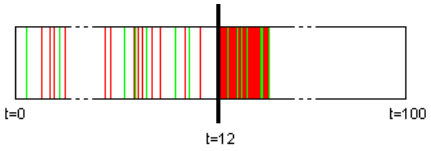

Figure 4.5: *Simplified demonstration of how test and training samples are taken on t=12, in which each red line represents a negative instance and each green line represents a positive instance.*

4.5 Implementations

In Chapter 3, two types of machine learning were identified; eager and lazy learning. As was discussed in that chapter, eager learners focus on the learning (or abstraction) phase, whereas lazy learners put their effort in the classification task. As an example of lazy learning, memory-based learning was mentioned as a classification method. For eager learning, rule induction was described. For this research, RIPPER was chosen as an implementation of eager learning classification. IB1 was chosen as a lazy learning classifier, using Timbl. Both implementations will be described below.

4.5.1 RIPPER

For the experiments conducted here, RIPPER was chosen as an implementation of an eager learning classifier. RIPPER is an abbreviation of Repeated Incremental Pruning to Produce Error Reduction (Cohen, 1995).

There are two reasons why RIPPER was chosen. First of all, it is an extremely efficient algorithm on large samples. It scales nearly linearly with the number of training examples. Secondly, it can efficiently process noisy datasets containing hundreds of thousands of examples. RIPPER is shown statistically to be a clear improvement over IREP. On noisy datasets, RIPPER is much more efficient than for instance C4.5rules (Cohen, 1995).

As was discussed in Section 4.2, the training used throughout this experiment may be quite noisy. Secondly, it gives error rates that are competitive with C4.5rules with respect to error rates (Cohen, 1995).

RIPPER splits the training set in two. On the basis of one half, it induces rules in a straightforward way (roughly by trying to maximise coverage for each rule), with potential over fitting. When the induced rules classify instances in the other half below a certain threshold, they are not stored. Rules are induced per class. By default the ordering is from low-frequency classes to high frequency ones, leaving the most frequent class as the default rule, which is generally beneficial for the size of the rule set (van den Bosch et al., 2001).

RIPPER produces hypothesis, based on the training data, that looks like:

```
y 55731   52  IF sourceip_boolincident = 1 .
y 1707    0   IF domainname_countseenday <= 1280 sourceip_countincident >= 1
                 protocol_timediff1 >= 1 sourceip_countseen <= 48
                 sourceip_countincident <= 1 .
y 138     3   IF asn_countseenmonth <= 1485 sourceip_countincident >= 1
                 protocol_timediff1 >= 1 sourceip_countseen <= 37
                 sourceip_countseen <= 11 .
n 59832 36 IF .
```

These rules that are induced are human-readable if-then rules, which can then be applied to the test set.

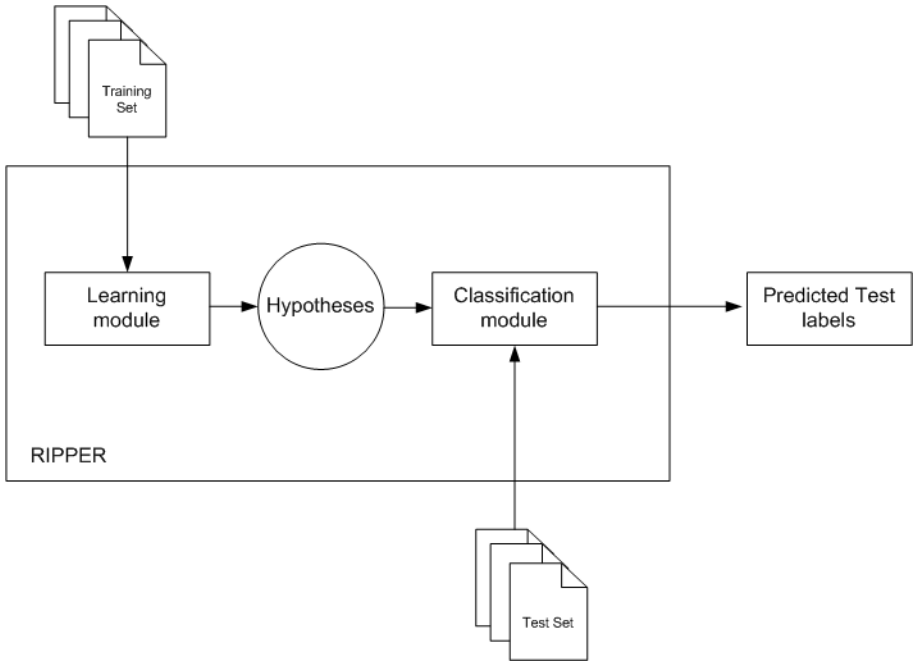

Figure 4.6: *Overview of classification with RIPPER*

4.5.2 Attribute selection

There are in general three reasons to use feature subset selection for machine learning. First of all, it can result in enhanced performance. Secondly, it provides a reduced hypothesis search space, and third of all, it reduces the requirements for storage equipment (Hall and Smith, 1996).

The purpose of feature selection is to decide which of the initial (possibly large number of) features to include in the final subset and which to ignore. If there are n possible features initially, then there are n^2 possible subsets. The only way to find the best subset would be to try them all, which is clearly prohibitive for all but a small number of initial features (Hall and Smith, 1996).

During this research, Weka[2] was used for attribute selection. Weka contains both Infogain and CFS selection methods. Attribute selection was only applied for feature selection on the training set for IB1. The training set of RIPPER was not reduced in the number of features. More or less indirectly rule induction already contains feature subset selection, by not selecting features in the generated rules.

[2] `http://www.cs.waikato.ac.nz/ml/weka/`

InfoGain

The first feature selection criterion that is used, is information gain of a feature. In order to define information gain precisely, a measure commonly used in information theory, called entropy, will be described first. Given a collection S, containing positive and negative example of some target concept, the entropy of S relative to this boolean classification is (Mitchell, 1997b):

$Entropy(S) = -p_{\oplus} \log(2) p_{\oplus} - p_{\ominus} \log(2) p_{\ominus}$

In general, for non-boolean features, the entropy of a feature can be measured by:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log(2) p_i$$

Where p_i is the proportion of S belonging to class i, and in which the target attribute can take on c different values (Mitchell, 1997b).

Information gain is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The information gain is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. Thus, the information gain, Gain(S,A) of an attribute A relative to a collection of examples S, is defined as (Mitchell, 1997b):

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where Values(A) is the set of all possible values for attribute A, and Sv is the subset of S for which attribute A has value v.

For this experiment, the features were ranked by InfoGain, and the first N features were selected. The N was chosen as 6, 8 and 10. The script for feature selection using InfoGain are taken in Chapter B.

CFS

The heuristic by which CFS measures "goodness" of feature subsets takes into account the usefulness of individual features for predicting the class label along with the level of intercorrelation among them. CFS evaluates and hence ranks feature subsets rather than individual features (Hall, 1998). The hypothesis on which the heuristic is base can be stated:

Good feature subsets contain features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other.

The algorithm CFS uses features' performances and intercorrelations to guide its search for a good subset of features. The correlation-based evaluation heuristic employed by CFS appears to choose feature subsets that are useful to the learning algorithms by improving their accuracy and making their results easier to understand (Hall and Smith, 1996).

Various heuristic search strategies such as hill climbing and Best First are often applied to search the feature subset space in reasonable time. CFS first calculates a matrix of feature-class and feature-feature correlations from the training data, and then searches the feature subset space using a best first search. The best first search starts with an empty set of features, and generates all possible single feature expansions. The subset with the highest evaluation is chosen and expanded in the same manner, by adding single features. If expanding a subset results in no improvement, the search drops back to the next unexpanded subset, and continues from there (Hall, 1998).

Given enough time a Best-First search will explore the entire search space, so it is common to limit the number of subsets expanded that result in no improvement (Hall and Smith, 1996). During this experiment, the Best First search method was used, using a search depth of 5. The script that was used to choose feature subsets, based on CFS, is shown in Appendix B.

4.5.3 IB1

IB1 is an implementation of the simplest similarity based learner known as nearest neighbour. IB1 simply finds the stored case closest (usually according to some Euclidean distance metric) to the instance to be classified. The new instance is assigned to the retrieved instance's class (Hall and Smith, 1996).

IB1 is identical to the nearest neighbour algorithm except that it normalises its attributes' ranges, processes instances incrementally and has a simple policy for tolerating missing values. The similarity of instances is calculated by the following formula:

$$Similarity(x, y) = -\sqrt{\sum_{i=1}^{n} f(x_i, y_i)}$$

where the instances are described y n attributes. The function $f(x_i, y_i)$ is defined as $(x_i - y_i)^2$ for numeric-valued attributes and $f(x_i, y_i) = (x_i y_i)$ for Boolean and symbolic-valued attributes. Missing attribute values are assumed to be maximally different from the value present. If they are both missing, then $f(x_i, y_i)$ yields 1 (Aha et al., 1991).

During this experiment, Timbl[3] was used, which contains an implementation of the IB1 algorithm.

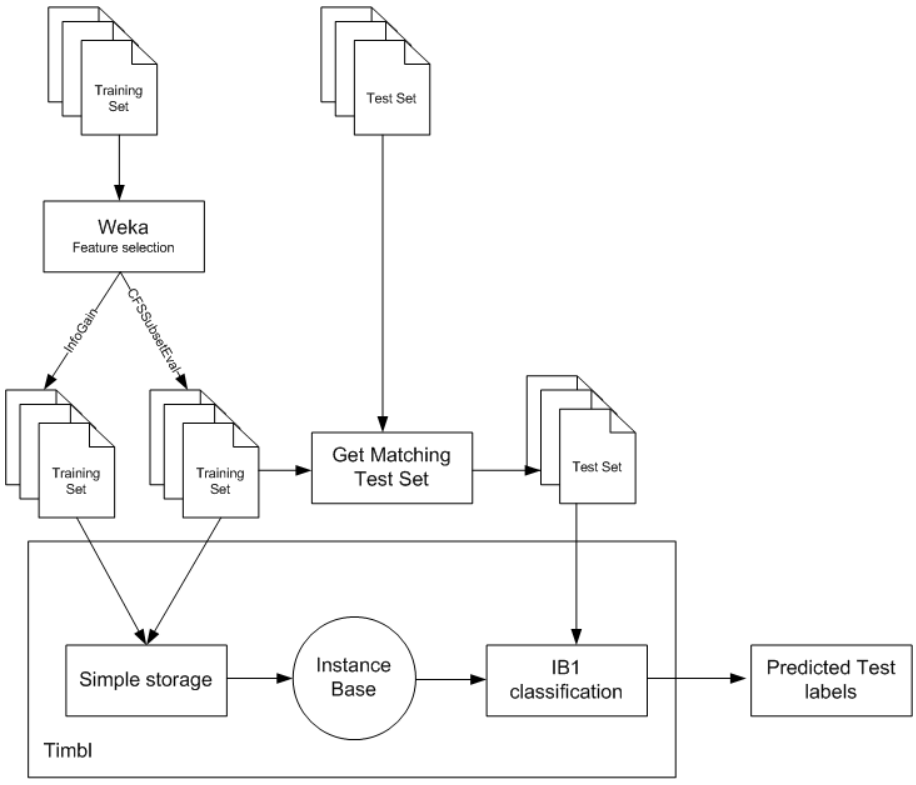

Figure 4.7: *Overview of the classification with Timbl adapted from (Hendrickx, 2005)*

4.5.4 Hybrids

For this research, three experiments were used with hybrid machine learners. First of all, Rules-R-H was applied to the training and data sets, containing all 224 features. It was impossible to apply Rules-A-H on all 224 features, because it was computationally too expensive. RIPPER was used in order to induce rules, based on all 224 features. The hypotheses created with RIPPER were

[3] `http://ilk.uvt.nl/timbl/`

than converted to binary feature vectors, as described in Section 3.5.1. These binary rule-features were than offered to Timbl, applying IB1 on these vectors.

As a second experiment, since it was computationally too expensive to apply Rules-A-H on the entire set of 224 features, the most informative features were selected using CFS. Having selected these features, they were than used to generate rules with RIPPER. The hypotheses that RIPPER produced, were then converted to binary feature vectors for each instance. Again, these converted binary features were given to Timbl to apply IB1 on them, thus classifying the instances with Rules-R-H.

Third of all, the same feature subset was used to create the same binary feature vectors. These were appended to the initial feature of each instance, instead of a replacing the initial features. Thus, a Rules-A-H hybrid was created.

All the experiments described above are summarised in Table 4.2. Note that the abbreviation "InfoGain-6" is used for the fact that InfoGain was used as a feature selector, with the selection of 6 features.

Feature selection method	**Classification method**
-	RIPPER
-	Rules-R-H
InfoGain-6	IB1
InfoGain-8	IB1
InfoGain-10	IB1
CFS	IB1
CFS	RIPPER
CFS	Rules-R-H
CFS	Rules-A-H

Table 4.2: *Summary of the combination of classification and feature selection methods used.*

4.6 Evaluation Methods

In order to evaluate the general performance of different classifiers, different metrics were used. These metrics were calculated using the confusion matrix, which shows the predicted and actual classifications. Since the number of classes is two in this experiment (either positive or negative), the size of the confusion matrix (λ x λ) is 2x2.

	Predicted Negative	**Predicted Positive**
Actual Negative	a	b
Actual Positive	c	d

Table 4.3: *Confusion Matrix*

As mentioned in Section 2.3.2, there are two types of errors that can be made; Type I and Type II errors. The number in "b" in the confusion matrix corresponds with the type I errors. The number in "c" equals the number of type II errors made. Both false positives and false negatives have to be reduced (Gowadia et al., 2005), which means that the accuracy has to be as close to 1 as possible. The accuracy is the proportion of the total number of predictions that were correct, and is measured by: $accuracy = \frac{a+d}{a+b+c+d}$

Furthermore, the precision is the proportion of the predicted positive cases that were correct. The precision of the classification is calculated by: $precision = \frac{d}{b+d}$.

Another interesting metric for evaluating classification methods is recall (or true positive rate), which is the proportion of alerts that were correctly identified, using the following equation: $recall = \frac{d}{c+d}$.

Finally, since the equation mentioned above for accuracy may not be an adequate performance measure, the F-Measure was used. Sparseness of the positive examples could cause the average classification accuracy on the testing set to be unreliable (Kubat and Matwin, 1997). In case there are 100 cases, 95 of which are negative cases, and 5 of which are positive cases. If the system classifies them all as negative, the accuracy would still be 95%, even though the classifier missed all the positive instances. For this reason, the weighted harmonic mean of precision and recall, with β given a value of 1, can be calculated as:

$$F = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * precision + recall}$$

The relationship between precision, recall and F-measure ($\beta = 1$) is shown in Figure 4.8. This figure illustrates that both a high recall and a high precision is necessary to get a F-measure close to 1. The line in this figure represents the F-measure. If either recall or precision is relatively low, this immediately effects the F-measure.

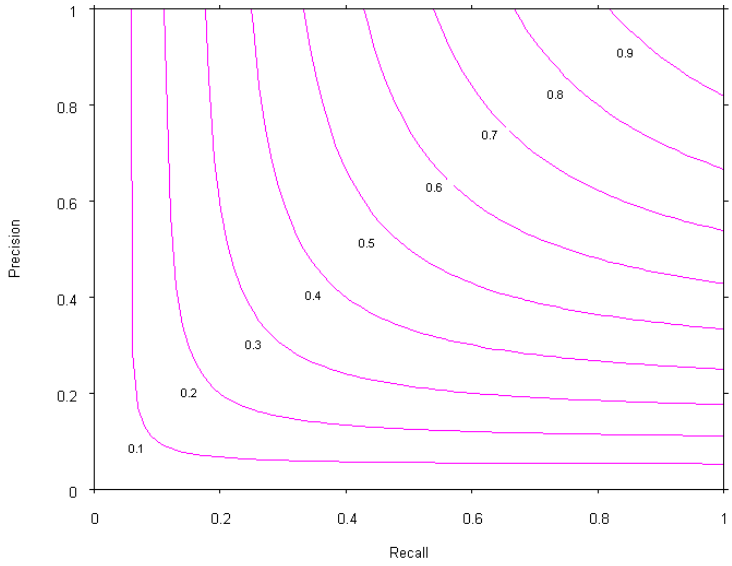

Figure 4.8: *Relationship between precision, recall and F-measure where* $\beta = 1$.

For each experiment, accuracy, precision, recall and F-measure were calculated in order to evaluate the classification. Having discussed the research method, the next chapter will discuss the measurements of the experiment, and will give an analysis of these measurements.

Chapter 5

Results

The results of the experiments, summarised in Table 4.2, will be this chapter's topic. First of all, the measurements will be given for the metrics discussed in Section 4.6 (i.e. precision, recall, accuracy and F-measure). This will be done in Section 5.1, and will only contain a brief statement of the measurements. After this summary of the measurements, the analysis of these result will take place in Section 5.2. This analysis phase will provide a detailed analysis on why these results were found. At the end of this chapter, some conclusions will be drawn based on the measurements and analysis.

5.1 Measurements

The results of the experiments will be presented in this section. The results will be divided into the results of RIPPER, the results of IB1 classification and the results of the hybrid machine learning algorithms.

The results of all the classification tasks are referred to in Appendix C. The results are measured over different test sets, as was described in Section 4.4.2. As was explained in that section, this was done in order to gain an insight in the period after which the machine learner should be re-trained. In the results however, the largest test set of 500 positive instances ahead, show better results in all cases than the smaller test sets. This is also shown in Figure 5.1. In general, a larger population in the test set gives a more realistic view on the results (Allison, 1999). Therefore, the measurements will be discussed for the largest test set (i.e. 500 positive instances ahead).

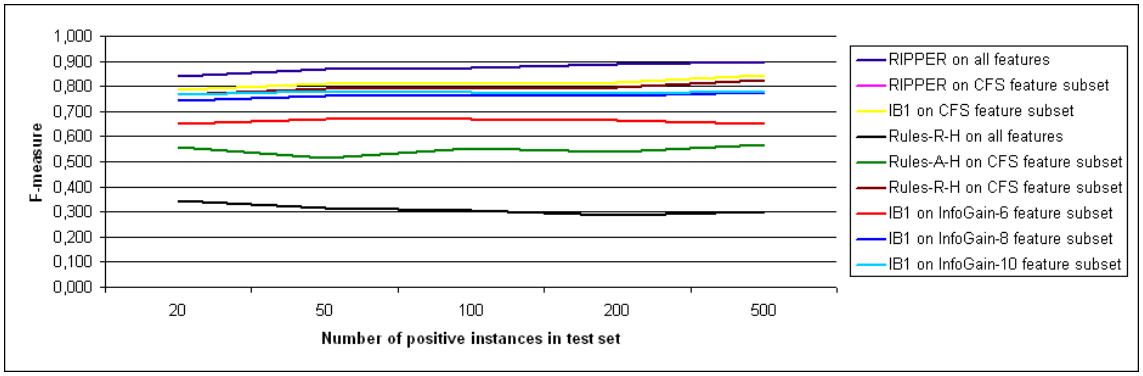

Figure 5.1: *In most cases, increasing the number of positive instances in the test set leads to a higher F-measures.*

5.1.1 Measurements of RIPPER

The measurements of rule induction algorithms for the classification task, as discussed in the previous chapter, are split in rule induction on the entire feature set, and rule induction on a feature subset (using CFS), which is also summarised in Table 4.2.

RIPPER on all features

The results of the classification with RIPPER on all features are referred to in Appendix C. The tables in the appendix are split for different sizes of test sets (20, 50, 100, 200 or 500 positive instances ahead).

When looking at 500 positive instances in the test set, the average precision measured over 50 points in time varies from 75.12% to 83.10% for different positive versus negative ratios in the training set, as shown in Table 5.1.

Ratio	0.5	1	2
Avg. Precision	75.12%	78.57%	83.10%
Avg. Recall	99.72%	99.62%	99.48%
Avg. Accuracy	94.40%	95.39%	96.59%
Avg. F-measure	0.845	0.872	0.900

Table 5.1: *Average metrics for RIPPER on all features.*

RIPPER on a subset of features using CFS

In contrast to running RIPPER on the entire set of features, RIPPER was also used to induce rules on data sets with features selected by CFS. The results of this experiment is summarised in Table 5.2. The location of the entire set of results can be found in Appendix C. The analysis of the results of RIPPER on all features and on a subset of features will be given in Section 5.2.1.

Ratio	0.5	1	2
Avg. Precision	57.46%	61.94%	74.55%
Avg. Recall	99.71%	98.67%	96.74%
Avg. Accuracy	85.52%	87.59%	92.10%
Avg. F-measure	0.714	0.744	0.826

Table 5.2: *Average metrics for RIPPER on a subset features using CFS.*

5.1.2 Measurements of IB1

As was shown in Table 4.2, there are four types of experiments that use IB1 classification. These experiments vary in the selection method of features, which is either InfoGain or CFS. For the selection method based on InfoGain, different data sets were made with 6, 8 and 10 features. The results of the IB1 classification task with different feature selection methods will be discussed in the next subsections.

IB1 on a feature subset of 6 features selected by InfoGain

Having selected 6 features out of the 224 features, based on their individual info gain, as was discussed in Section 4.5.2, the IB1 classification was evaluated. The results of this classification task are referred to in Appendix C. A summary of the evaluation is shown in Table 5.3.

Ratio	0.5	1	2

Avg. Precision	49.54%	51.56%	53.21%
Avg. Recall	99.57%	99.40%	99.01%
Avg. Accuracy	81.33%	82.83%	83.96%
Avg. F-measure	0.645	0.663	0.649

Table 5.3: *Average metrics for IB1 on a feature subset containing 6 features selected by InfoGain.*

IB1 on a feature subset of 8 features selected by InfoGain

In order to demonstrate the effect of adding features to the 6 selected features above, 8 features were selected based on their InfoGain. The location of all the results of IB1 classification on this expanded feature set can be found in Appendix C. Moreover, a summary of the results is shown in Table 5.4.

Ratio	0.5	1	2
Avg. Precision	50.30%	51.86%	65.58%
Avg. Recall	99.58%	99.41%	99.15%
Avg. Accuracy	81.74%	83.04%	90.77%
Avg. F-measure	0.652	0.666	0.775

Table 5.4: *Average metrics for IB1 on a feature subset containing 8 features selected by InfoGain.*

IB1 on a feature subset of 10 features selected by InfoGain

To illustrate the effect of further increasing the number of selected features, 10 features were selected from the 224 results, based on their InfoGain. The location of the results of the IB1 classification that was applied to the sets with 10 features is shown in Appendix C. Furthermore, a summary of these results is shown in Figure 5.5.

Ratio	0.5	1	2
Avg. Precision	50.67%	52.28%	66.69%
Avg. Recall	99.57%	99.40%	99.13%
Avg. Accuracy	81.99%	83.26%	91.07%
Avg. F-measure	0.655	0.669	0.783

Table 5.5: *Average metrics for IB1 on a feature subset containing 10 features selected by InfoGain.*

IB1 on a feature subset of features using CFS

In stead of selecting features based on their InfoGain, another experiment was set up in which features were selected using CFS, as was discussed in Section 4.5.2. Subsequently, IB1 classification was performed. The location of the results of this classification is shown in Appendix C. A summary of these results is shown in Table 5.6.

Ratio	0.5	1	2
Avg. Precision	59.39%	64.57%	76.75%
Avg. Recall	99.09%	98.36%	85.37%
Avg. Accuracy	85.37%	88.43%	93.06%
Avg. F-measure	0.728	0.765	0.843

Table 5.6: *Average metrics for IB1 on a feature subset containing features selected by CFS.*

5.1.3 Measurements of Hybrids

As was described in Section 4.5.4, three experiments were conducted with hybrid machine learners. First of all, Rules-R-H was applied to all 224 features. Furthermore, both Rules-R-H and Rules-A-H were applied to feature subsets, having used CFS for feature selection. The results of these experiments will be described in this section.

Rules-R-H on all features

The results of applying Rules-R-H on the entire set of features, resulted in the measurements referred to in Appendix C. A summary of these measurements is in Table 5.7. An analysis of these results will be given in Section 5.2.3.

Ratio	0.5	1	2
Avg. Precision	28.64%	31.35%	30.77%
Avg. Recall	87.08%	64.52%	55.09%
Avg. Accuracy	38.14%	51.91%	53.92%
Avg. F-measure	0.365	0.360	0.343

Table 5.7: *Average metrics for RIPPER on a subset features using CFS.*

Rules-R-H on a subset of features using CFS

The results of the application of Rules-R-H on a feature subset can be found on the website, referred to in Appendix C. A summary of these results is shown in Table 5.8. The results will be discussed in more detail in Section 5.2.3.

Ratio	0.5	1	2
Avg. Precision	57.95%	61.96%	74.58%
Avg. Recall	99.68%	98.66%	96.73%
Avg. Accuracy	85.69%	87.60%	90.98%
Avg. F-measure	0.717	0.744	0.826

Table 5.8: *Average metrics for RIPPER on a subset features using CFS.*

Rules-A-H on a subset of features using CFS

Using the Rules-A-H classification method as described in Section 4.5.4, on a feature subset using CFS, results in the measurements referred to in Appendix C. A summary of these results is shown in Table 5.9. An analysis of these results will be given in Section 5.2.3.

Ratio	0.5	1	2
Avg. Precision	71.46%	29.55%	47.89%
Avg. Recall	93.32%	98.99%	97.23%
Avg. Accuracy	74.55%	31.58%	51.52%
Avg. F-measure	0.731	0.401	0.567

Table 5.9: *Average metrics for RIPPER on a subset features using CFS.*

5.2 Analysis

Ideally, an IDS has a relatively high recall, which means that all alerts that are incidents are identified by the IDS as incidents. A relatively high precision is a second requirement, in order to be effective. For each classification method, an analysis will be given of the results discussed in the previous chapter.

5.2.1 Analysis of RIPPER

As results of other experiments have demonstrated, the precision of RIPPER is higher than the precision of IB1. This is due to the fact that the set of rules seeks to capture the specificity of the minority class (Hoste, 2005). As the results show in Section 5.1.1, this is also true for these

experiments. RIPPER has a relatively high precision compared to IB1, in case the proportion of the negative class is bigger than the proportion of the positive class ($p_{\ominus} > p_{\oplus}$). This is also illustrated by Figure 5.2 and Figure 5.3.

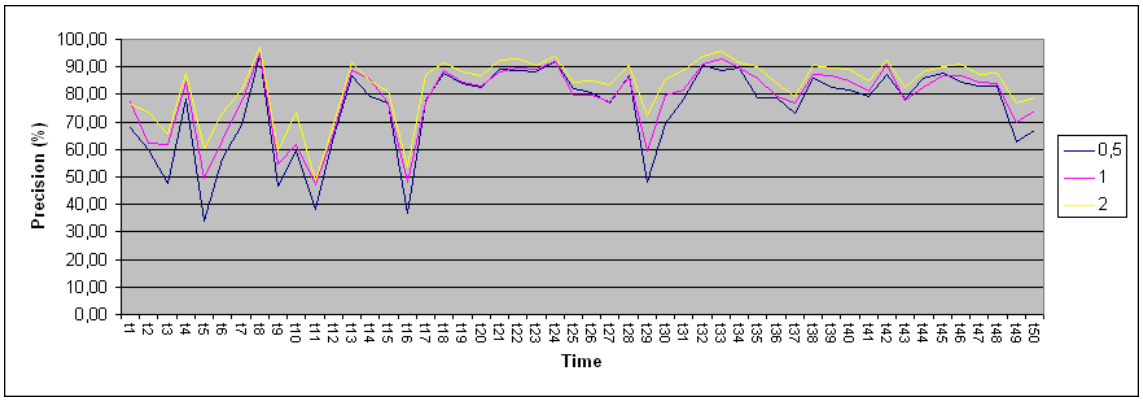

Figure 5.2: *Precision of RIPPER over time with negative:positive ratios of 0.5:1, 1:1 and 2:1.*

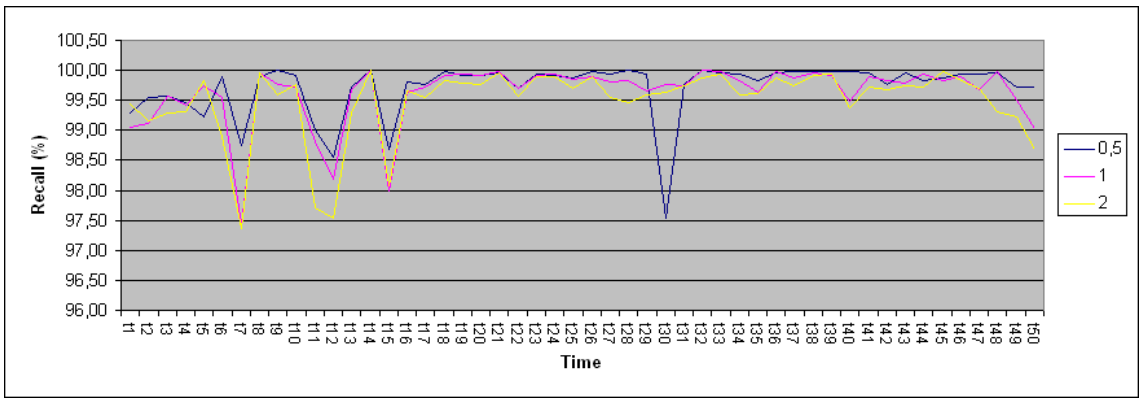

Figure 5.3: *Recall of RIPPER over time with negative:positive ratios of 0.5:1, 1:1 and 2:1.*

Depending on the exact tuning of the IDS, one may choose for a negative:positive ratio in the training set. As shown by Figure 5.2, the precision of the classification task improves with a higher ratio. However, the recall suffers from the increase of the number of negative instances.

Applying RIPPER on a subset of features that were selected by CFS does not improve the performance of RIPPER. This can be explained by the fact that RIPPER selects its features by itself, by taking certain features into its induced rules, or by leaving them out. Making a feature subset in advance will decrease the search space of RIPPER, thus decreasing the performance.

An example of a rule set produced by RIPPER is shown below.

```
Rule 1: y 7183  6 IF sourceip_boolincident = 1 .
Rule 2: y 35    0 IF domainname_countseenhour <= 254 sourceip_countincident >= 1
                     destport_boolincident = 1 asn_countseen <= 1.769e+06 .
Rule 3: y 165   2 IF bgp_countseenhour <= 254 sourceip_countincident >= 1
                     sourceip_countseen <= 43 asn_countseen <= 28420 .
Rule 4: y 103   7 IF domainname_countseenday <= 1280 sourceip_countincident >= 1
                     timestamp <= 1.13197e+09 asn_timediff1 >= 1 .
Rule 5: y 6     0 IF asn_countseen >= 1.76891e+06 sourceip_countincident <= 1 .
Rule 6: y 3     1 IF asn_countincident >= 1543 .
Rule 7: y 3     0 IF sourceipid = 12438 .
Rule 8: n 80750 1 IF .
```

This sample is taken at the beginning of the time line (t=2). Several interesting things can be observed from this rule set. Rule 1 indicates that if the last alert of this ip address was an incident, this alert will be labelled as an incident as well. This first rule is a strong predictor, classifying 7183 instances correctly, and only 6 incorrectly. Furthermore, this particular rule appears in the first rule of every rule set from t=1 to t=50.

Rule 2 to rule 6 contain (compound) statements with numeric features. Although an analyst probably did not count whether the AS number appeared more than 1543 times in incidents, the analyst may have had the feeling that the AS number was involved more often in an incident. This feeling is translated in explicit rules by RIPPER.

Another interesting rule is rule 7. Apparently, this source ip address caused an incident, without the alerts of it being triggered by any other rule. In this case, RIPPER creates an explicit rule for this ip address.

RIPPER closes the rule set by labelling all the other instances that do not match with one of the rules as a non-incident by using rule 8. This can be explained by the fact that a negative versus positive ratio of 2:1 is used. Therefore, the majority class in the training set is a non-incident.

Concluding from previous observations, RIPPER seems to be a good classifier for this classification task. RIPPER was able to discover knowledge from the training sets, in an efficient way considering the results. The rules RIPPER produces are intelligible for human beings, as was demonstrated above. Moreover, these rules make the information hidden in the training data explicit, thus providing discovered knowledge.

5.2.2 Analysis of IB1

In other research, the k-NN IB1 showed a relatively high recall compared to rule induction (e.g. (Hoste, 2005)). In this research, however, the recall of RIPPER is already relatively high. Even though IB1 has an average recall of 99% in some cases, the results are not higher than the recall of RIPPER.

Another observation in other research is that the noise-tolerant k-nearest neighbour algorithm performs poorly when applied to small (insufficiently sized) training sets (Aha et al., 1991). As can be concluded from Figure 5.4, the weighted harmonic mean of precision and recall increases over time, when more training data is provided to the IB1 classifier. Furthermore, also illustrated by Figure 5.4, a higher negative:positive ratio also leads to a higher F-measure. This supports the statement that IB1 classification performs better when more training data is provided to it.

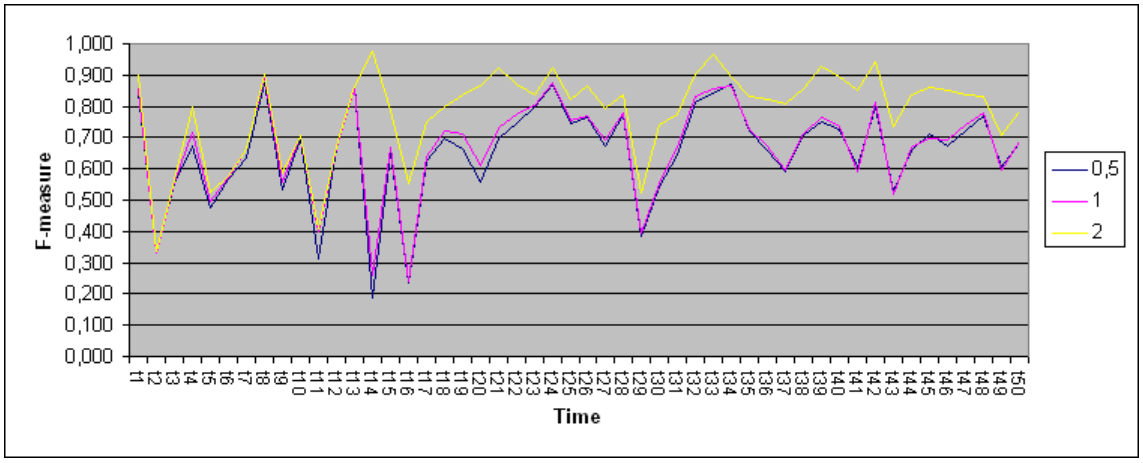

Figure 5.4: *F-measure of IB1 with 10 selected features (based by InfoGain) over time with negative:positive ratios of 0.5:1, 1:1 and 2:1.*

When using attribute selection for the IB1 implementation based on the information gain of the features, the higher the number of features selected (N=10), the higher the F-measure is compared

to a lower value for N (e.g. N=6). This is mainly due to the fact that the precision is elevated by a larger number of features in the subset. This is also shown in Figure 5.5. The differences between the selection of 8 and 10 features are relatively small however.

The training sets that were generated using CFS feature selection lead to a higher precision with IB1 classification, than the ones based on InfoGain. The reason for this is that the correlation between features is not taken into account by the ranking of features based on information gain. Earlier experimental results show promising results for CFS as a practical feature selector for common machine learning algorithms (Hall and Smith, 1996), such as IB1.

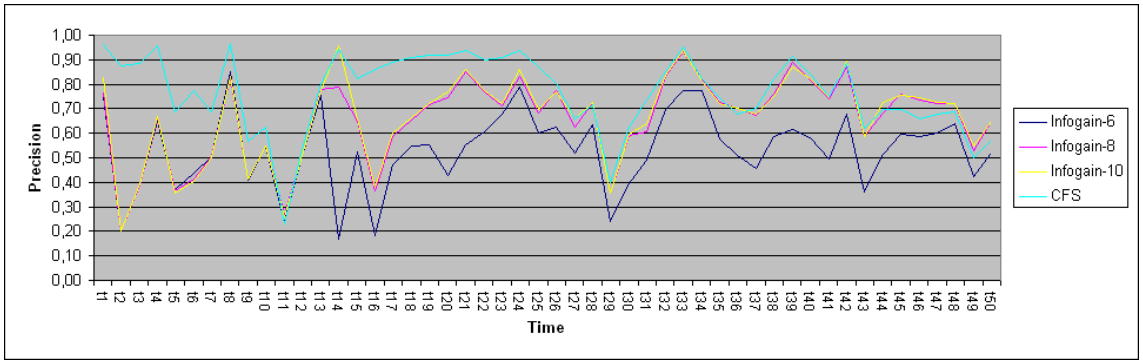

Figure 5.5: *Precision of IB1 using CFS or InfoGain selecting 6, 8 or 10 features with ratio 2:1.*

Above-mentioned observations lead to the conclusion that IB1 is not a very good classifier for this classification task. Although a higher positive versus negative ratio in the training sets, the selection of more features or by selecting features using CFS, enhance the result, RIPPER performs better on both recall and precision. One of the reasons why IB1 does not not score as well as RIPPER may be that RIPPER is able to ignore features. RIPPER can ignore a feature by simply not taking it into one of its rules, whereas IB1 always assigns a certain weight to a feature. The ability to disable features that have a negative influence on the outcome may improve the results.

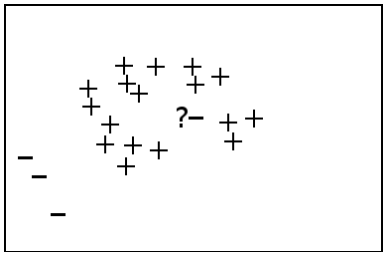

Figure 5.6: *Example of a potential problem with IB1 classification.*

Another reason why IB1 is not a very good classifier for this classification task can be clarified by Figure 5.6, which depicts a number of positive instances (denoted with "+"), and negative instances (denoted with "-"). When a new test instance ("?") is classified, the closest neighbour of the training set is used to determine its class. In case of noisy data, however, it may happen that a wrong label is assigned, as shown in Figure 5.6, since IB1 only uses the closest neighbour in the training set. This makes IB1 susceptible to noise. The performance of k-NN may be improved by increasing the k. Looking at the labels of the 3 closest neighbours, for instance, may improve the results.

5.2.3 Analysis of Hybrids

Based on earlier case studies ((van den Bosch, 2004) and (Hendrickx and van den Bosch, 2005)), the combination of eager and lazy classification is expected to improve the results of the classification

task. The results, however, show that for this particular classification task, hybrids do not perform better than the form of eager learning used in this research.

Rules-R-H applied on all 224 features is the worst classifier that was used for this task. Figure 5.8 shows the precision of Rules-R-H for different ratios. Surprisingly enough, providing more training data to the classifier lowers the recall, as demonstrated by Figure 5.8. The recall for the classification task with a majority of positive instances is better than the case in which negative instances form the majority of the training set.

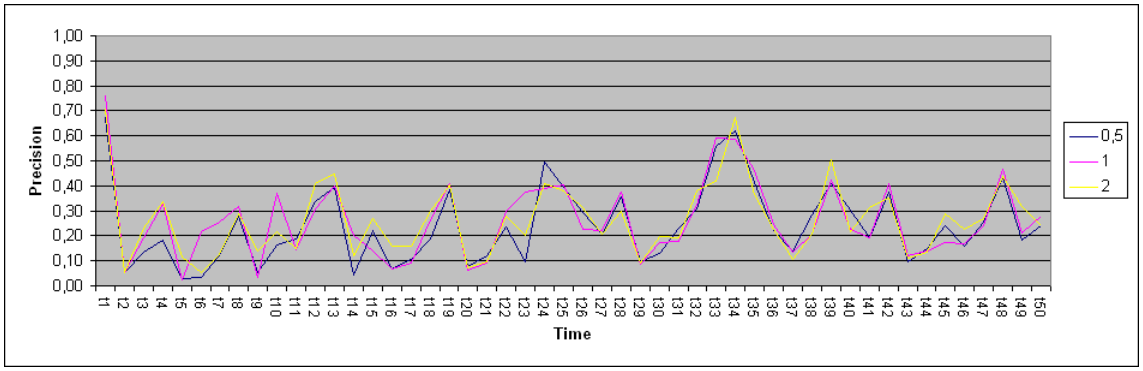

Figure 5.7: *Precision over time of Rules-R-H on all features, for different ratios.*

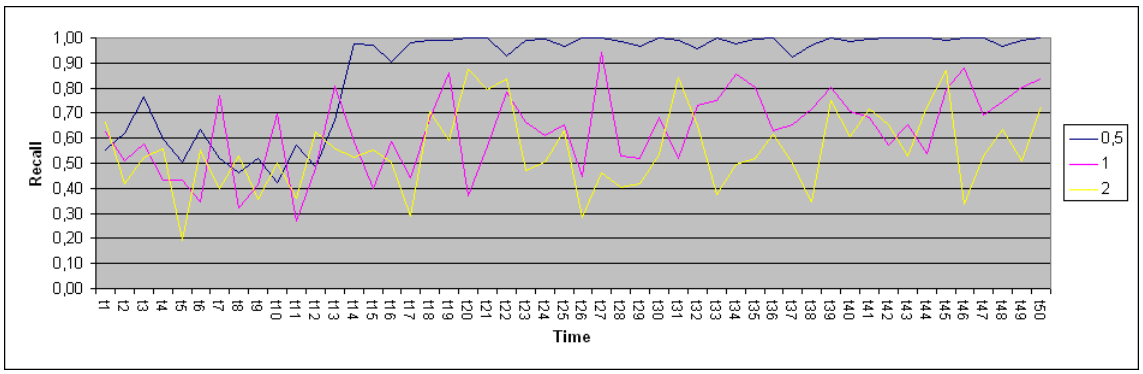

Figure 5.8: *Recall over time of Rules-R-H on all features, for different ratios.*

Selecting features based on CFS before applying Rules-R-H improves the performance to almost exactly the same level (in recall, precision and F-measure) as RIPPER on a feature subset using CFS.

The precision and recall of Rules-A-H over time for different ratios is shown in Figure 5.9 and Figure 5.10 respectively. As shown by these figures, Rules-A-H suffers from the same syndrome as Rules-R-H on the entire feature set; the classification method performs better when the positive instances form the majority class.

The reason why the hybrids proposed in this thesis do not perform as well as anticipated, may be that IB1 classification does not score well on this classification task. The potential reasons for this were already mentioned in Section 5.2.3. Both Rules-R-H as Rules-A-H use IB1 in order to search for the nearest neighbour in the training set.

5.3 Conclusions

In order to be effective, false positives and false negatives have to be reduced. Of these two types of errors, the false negatives are more problematic for an IDS, because it doesn't identify an alert as

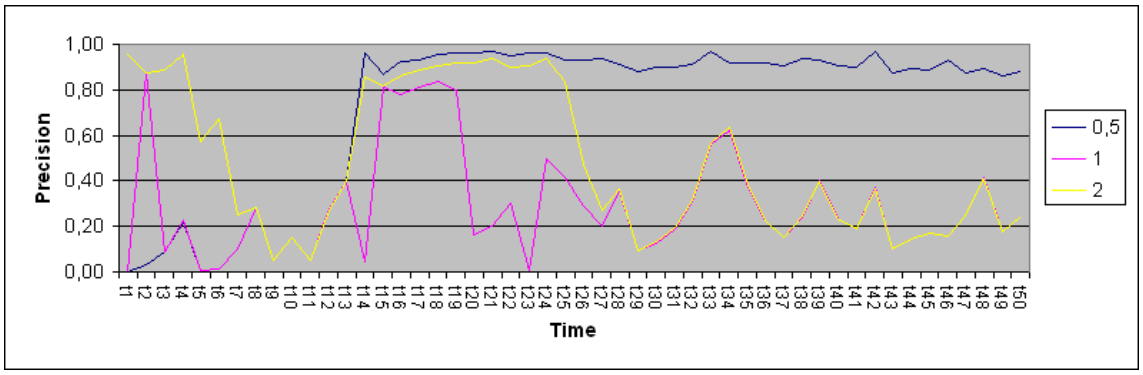

Figure 5.9: *Precision over time of Rules-R-H on a feature subset, for different ratios.*

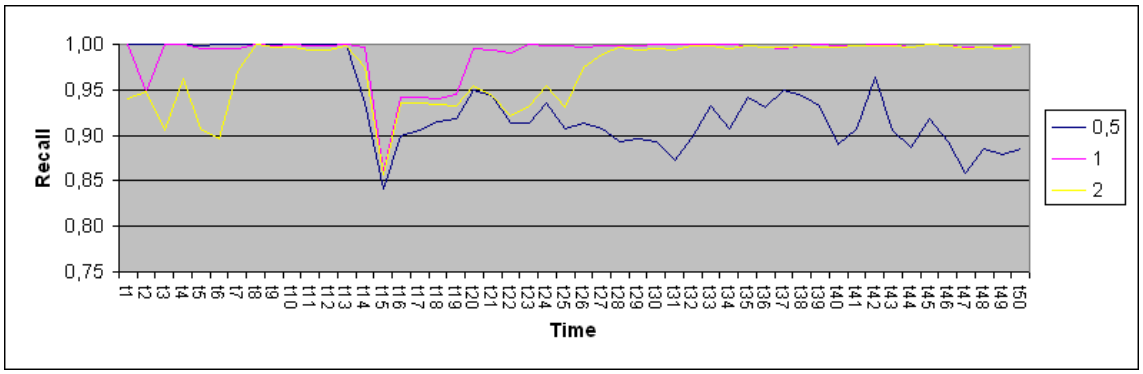

Figure 5.10: *Recall over time of Rules-A-H on a feature subset, for different ratios.*

an incident, and lets it pass through the filtering procedure. Therefore, recall is the most important metric, followed by precision.

In this situation, rule induction seems highly effective as a classification method, with a recall of 99.48%, and a precision of 83.10%, on a "negative:positive ratio" of 2. The rules RIPPER produces are understandable for human beings, and make the discovered knowledge explicit, thus producing knowledge. RIPPER selects its own features by either taking them into its rule-set, or by leaving it out. Therefore RIPPER does not need feature selection. This is also proven by the experiment that uses RIPPER as a classification method and CFS as feature selection method, which does not improve the results.

Classification based on IB1 performs not as well as RIPPER. With an increase of the number of features used for this classification the measurements improve, though not enough to outperform RIPPER. Using CFS as a feature selection method results in a higher precision, but also lowers the recall. The reasons why IB1 does not perform well, may be due to the fact that features can not be totally disabled. Another reason could be that IB1 only looks at the closest neighbour in the training set. Increasing the k to (for instance) 3 may improve the performance of the k-NN algorithm.

The hybrids, combining eager and lazy learning, score rather disappointing. This may be due to the fact that IB1 on itself does not perform as well as RIPPER. This may be caused by the same reasons why IB1 does not perform very well. Most promising results for the hybrids are provided by Rules-A-H, on a training set in which the positive instances form a majority class.

Over time, it may be necessary to retrain the machine learners in order to keep them up-to-date. During these experiments, however, a decrease of recall or precision was not found. Varying the number of positive instances in the test set showed no effect on these metrics.

Hence, UvT-CERT could at first use the rules induced by RIPPER as mentioned in Section 5.1.1

in order to have an effective filtering process. This filter may work as an indication for UvT-CERT members. If the members do not agree with the label presented by the filter, they may change the label and may retrain the filter after some time to regain its effectiveness.

Chapter 6

Conclusion

6.1 Results

In order to answer the main research question of this thesis, subquestions were formulated in section 1.1. Before answering the main research question, these subquestions will be answered first.

What is an Intrusion Detection System?
IDSs may complement other preventive controls (e.g. firewalls) as the next line of defence within the organisation (Pfleeger and Pfleeger, 2003). An IDS is a device that is placed inside a protected network to monitor what occurs within the network. The goal of intrusion detection systems is to accurately detect anomalous network behaviour or misuse of resources, sort out true attacks from false alarms, and notify network administrators of the activity. Many organisations now use intrusion detection systems to help them determine if their systems have been compromised (Carnegie Mellon University, 2001).

What is a computer security alert, what is a computer security incident and what are the differences between them?
A distinction is made between alerts and incidents by an intrusion detection system. Alerts are defined as all the observable actions on the computer network that are picked up by the sensors of an intrusion detection system. All alerts have value, but some have greater value than others (Bejtlich, 2004). The alert that has a high enough value, and which is considered to be a "security-relevant system event in which the system's security policy is disobeyed or otherwise breached (Shirey, 2000) is considered to be an incident.

What kind of different types of machine learning techniques exist?
First of all, machine learning can be divided in two categories; supervised and unsupervised machine learning algorithms (Hendrickx, 2005). In supervised learning, the input of the learning algorithm consists of examples (in the form of feature vectors) with a label assigned to them. The objective of supervised learning is to learn to assign correct labels to new unseen examples of the same task. In this thesis, the focus was on supervised learning algorithms, since the labels of historical incidents were known. Another distinction between types of machine learning is the one between eager versus lazy learning. Eager learning algorithms invest most of their effort in the learning phase, lazy learners defer the decision of how to generalise beyond the training data until each new query instance is encountered. Hybrids are mixtures of the eager and lazy learners. The reason for constructing hybrids is the contrast between memory-based learning and eager learning. Combining eager and lazy learners into hybrids, will produce machine learners that put effort in both the learning phase and the classification phase. This leads to the expectation that this double effort will be repaid with improved performance.

Which technique fits best in the situation of incident determination?
A high recall of the machine learner is of the utmost importance in incident recognition, since an IDS should not miss any incidents. Rule induction, which is a form of eager learning, performs

best in this situation, by generalising over the training sets. With the rules induced by the learner, new unseen test instances can be classified with relatively high certainty.

Is machine learning an effective tool to determine computer security incidents based on IDS input?
With a precision of 83.10% and a recall of 99.48%, rule induction seems to be highly effective to predict incident labels, based on labelled training instances. The rules induced by RIPPER could be applied in this particular situation, and the eager learner could be retrained if needed.

Does machine learning offer genuine knowledge discovery that is intelligible for human beings?
By using rule induction, implicit knowledge is extracted from training sets, and is made visible. The rules generated by RIPPER are if-then rules, which are understandable and readable for human beings. The rules make the information hidden in the training data explicit, thus providing discovered knowledge. Furthermore, this set of rules can be implemented relatively straightforward in IDS applications.

"Is it possible to distinguish computer security alerts from computer security incidents by using Intrusion Detection System logging and machine learning techniques, and what would be an appropriate technique?".
It is demonstrated by several experiments in this thesis that rule induction provides the best results in the classification of computer security alerts. With a recall of 99.48% in the experiments, this form of eager learning identifies almost all incidents as incidents. Moreover, the number of false alarms is the lowest measured in the experiments.

6.2 Recommendations for Future Work

Machine learning is a relatively new, and constantly changing field of science. Therefore, new techniques, implementations, and parameters should be evaluated for future use. Next to this constant movement, there are numerous recommendations for future work on the implementation as described in this paper.

First of all, during this research, individual lines of information were classified. It may be an improvement to the currently used strategy to inspect a sequence of lines, in stead of each line individually. A drawback for the newly proposed approach, could be that it is extremely difficult to point out a beginning and an end of a sequence of alerts.

Second of all, the parameters of the machine learners were not optimised. It appeared to be computationally too expensive to optimise the parameters of the machine learners before each test run. In future work, when more time is available to conduct these experiments, it could improve the machine learner's performance to use paramsearch[1] to optimise the parameters of machine learners.

Another way to further improve the performance of the machine learners, would be to train them with a higher ratio of negative instances. Since it was not within the scope of this research to find the optimal "positive-negative-ratio", it could enhance the performance of the machine learners. So the third recommendation would be too evaluate this ratio.

In the future, in order for this research to become more tangible, the filter of section 5.3 can be implemented relatively easily by taking the rules induced by RIPPER. This would be a first step in the direction of an adaptive application of a learning filter. After some time, retraining may be required. As a second step it is recommended to automate the process of retraining, thus taking advantage of the future possibilities machine learning has.

6.3 Practical Application

The efforts that were made during the research phase of this thesis can be implemented relatively easily in practical applications, such as AIRT. Sensors of an intrusion detection system, hidden in a

[1] `http://ilk.uvt.nl/~antalb/paramsearch/`

large enterprise network, can push all the alerts found on that network into a database. Additional features may be calculated for each of these alerts. By using RIPPER as a knowledge extractor, rules can be induced based on the labels provided by analysts. These rules can be implemented relatively straightforward.

The machine learner may be trained like a spam filter. The rules of RIPPER classify the alert either as a non-incident. Since RIPPER has a recall close to 100%, alerts that are an incident will hardly slip through the classifier. The analyst's job is to look at the outcome of the classification task, and he/she will have to decide on whether the filter was right or wrong. This pre-classification of alerts may save the analyst a lot of work. The work of the analyst is reduced to correcting possible errors of the machine learner. Moreover, by correcting the classification of the alerts, feed-back is given to RIPPER, once it is retrained, which will lead to a personally trained incident filter.

Bibliography

Aha, D. (1997). Lazy learning. In *Artificial Intelligence*, volume 11, pages 7–10. Kluwer Academic Publishers.

Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

Allison, P. (1999). *Multiple Regression: A Primer*. Prine Forge Press.

American Registry for Internet Numbers (ARIN) (2006). *AS Number registration*. `http://www.arin.net/registration/asn/index.html`. Last visit: 10-01-2006.

BBC News (2004). *Computer Hacking 'costs billions'*. BBC News, `http://news.bbc.co.uk/1/hi/business/3663333.stm`. Last visit: 05-29-2006.

Bejtlich, R. (2004). *The Tao of network security monitoring*. Addison Wesley.

Berk, V., Gray, R., and Bakos, G. (2000). Using sensor networks and data fusion for early detection of active worms.

Botha, M. and von Solms, R. (2003). Utilizing fuzzy logic and trend analysis for effective intrusion detection. In *Computers & Security*, volume 22, pages 423–434.

Carnegie Mellon University (2001). *Intrusion Detection Systems*. Carnegie Mellon University, `http://www.sei.cmu.edu/news-at-sei/features/2001/1q01/pdf/feature-3-1q01.pdf`. Last visit: 09-13-2006.

Cohen, W. (1995). Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*.

Daelemans, W., van den Bosch, A., and Zavrel, J. (1999). Forgetting exceptions is harmful in language learning. In *Machine Learning*, volume 34, pages 11–41.

Daelemans, W., Zavrel, J., van der Sloot, K., and van den Bosch, A. (2005). *TiMBL: Tilburg Memory-Based Learning - Reference Guide*. Induction of Linguistic Knowledge, `http://ilk.uvt.nl/downloads/pub/papers/ilk0501.pdf`. Last visit: 09-18-2006.

Ghahramani, Z. (2004). *Unsupervised Learning*. University College London.

Gowadia, V., Farkas, C., and Valtorta, M. (2005). Paid: A probabilistic agent-based intrusion detection system. In *Computers & Security*, volume 24, pages 529–545.

Hall, M. (1998). *Correlation-based Feature Selection for Machine Learning*. Waikato University.

Hall, M. and Smith, L. (1996). Practical feature subset selection for machine learning. In *Proceedings of the Australian Computer Science Conference (University of Western Australia)*. University of Waikato.

Hawkinson, J. (1996). *RFC 1930 - Guidelines for creation, selection, and registration of an Autonomous System (AS)*. BBN Planet, `http://www.ietf.org/rfc/rfc1930.txt`. Last visit: 09-11-2006.

Hendrickx, I. (2005). *Local Classification and Global Estimation*. Koninklijke drukkerij Broese & Peereboom.

Hendrickx, I. and van den Bosch, A. (2005). *Hybrid Algorithms with Instance-based Classification*. ILK, Tilburg University, `http://ilk.uvt.nl/~iris//Pubs/ecml_hybrids_final.pdf`. Last visit: 08-30-2006.

Hoste, V. (2005). *Optimization Issues in Machine Learning of Coreference Resolution*. Universiteit Antwerpen.

Huang, M., Jasper, R., and Wicks, T. (1999). A large scale distributed intruion detection framework based on attack strategy analysis. In *Computer Networks*, volume 31, pages 2465–2475.

Jackson, T., Levine, J., Grizzard, J., and Owen, H. (2004). An investigation of a compromised host on a honeynet being used to increase the security of a large enterprise network. In *Proceedings of the 2004 IEEE Workshop on Information Assurance and Security*. IEEE.

Krasser, S., Grizzard, J., Owen, H., and Levine, J. (2005). The use of honeynets to increase computer network security and user awareness. In *Journal of Security Education*, volume 1, pages 23–37.

Kubat, M. and Matwin, S. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *Proceedings of the 14th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann.

May, S. and Dornseif, M. (2004). Modelling the costs and benefits of honeynets. In *Procceedings of the 3th Annual Workshop on Economics and Information Security (WEIS04)*, `http://pi1.informatik.uni-mannheim.de/publications/show/64`.

Mitchell, T. (1997a). Does machine learning really work? In *AI Magazine*, pages 11–20.

Mitchell, T. (1997b). *Machine Learning*. McGraw-Hill International Editions.

Nilsson, N. (1996). *Introduction to Machine Learning*. Stanford University, `http://ai.stanford.edu/~nilsson/MLDraftBook/MLBOOK.pdf`. Last visit: 08-30-2006.

Pfleeger, C. and Pfleeger, S. (2003). *Security in computing*. Prentice Hall.

Pietraszek, T. (2004). Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection*, volume 3224/2004, pages 102–124.

Proctor, P. (2001). *The practical Intrusion Detection Handbook*. Prentice Hall.

Provos, N. (2004). *Honeyd Frequently Asked Questions*. `http://www.honeyd.org/faq.php`. Last visit: 09-18-2006.

Rekhter, Y. (1995). *RFC 1771 - A Border Gateway Protocol 4 (BGP-4)*. Cisco Systems, `http://www.ietf.org/rfc/rfc1771.txt`. Last visit: 09-11-2006.

Rietta, F. (2006). Application layer intrusion detection for sql injection. In *Proceedings of the 2006 ACM Syposium of Applied Computing (ACMSE-2006)*, pages 531–536, `http://www.rietta.com/papers/rietta_acmse2006.pdf`.

Shirey, R. (2000). *RFC 2828 - Internet Security Glossary*. GTE / BBN Technologies, `http://www.tilburguniversity.nl/services/its/security/uvt-cert/`. Last visit: 09-11-2006.

Siraj, A., Vaughn, R., and Bridges, S. (2004). Intrusion sensor data fusion in an intelligent intrusion detection system architecture. In *Proceedings of the 37th Hawaii International Conference on System Sciences*.

Staniford-Chen, S., Tung, B., Porras, P., Kahn, C., Schnackenberg, D., Feiertag, R., and Stillman, M. (1998). *The Common Intrusion Detection Framework - Data Formats*. IETF, `http://mirrors.isc.org/pub/www.watersprings.org/pub/id/draft-staniford-cidf-data-formats-00.txt`. Last visit: 09-13-2006.

Stillerman, M., Marceau, C., and Stillman, M. (1999). Intrusion detection for distributed applications. In *Communications of the ACM*, volume 42, pages 62–69.

Tesink, S. (2005). *Improving CSIRT Communication Through Standardized and Secured Information Exchange*. `http://www.tesink.org/thesis.pdf`. Last visit: 08-30-2006.

The Honeynet Project (2002). *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Addison-Wesley.

van den Bosch, A. (2004). Feature transformation through rule induction, a case study with the k-nn classifier. In *Proceedings on the workshop on advances in Inductive rule learning ath the ECML/PKDD 2004*, pages 1–15.

van den Bosch, A. and Daelemans, W. (1998). Do not forget: full memory in memory-based learning of word pronunciation. In *NeMLaP3/CoNLL98*, pages 195–204.

van den Bosch, A., Krahmer, E., and Swerts, M. (2001). Detecting problematic turns in human-machine interactions: Rule-induction versus memory-based learning approaches. In *Meeting of the Association for Computational Linguistics*, pages 499–506.

Varine, B. (2001). *Intrusion Detection FAQ: Should we outsource monitoring?* SANS Institute, `http://www.sans.org/resources/idfaq/outsource.php`. Last visit: 08-14-2006.

West-Brown, M. J., Stikvoort, D., Kossakowski, K.-P., Killcrece, G., Ruefle, R., and Zajicek, M. (2003). *Handbook for Computer Security Incident Rsponse Teams (CSIRTs)*. Carnegie Mellon Software Engineering Institute, `http://www.sei.cmu.edu/pub/documents/03.reports/pdf/03hb002.pdf`, 2 edition. Last visit: 17-02-2007.

Appendices

Appendix A

Features

The features used for this thesis can be divided into direct features, indirect features and calculated features. The direct features are shown in Table A.1. The indirect features are shown in Table A.2. For the features “asn”, “BGP”, “CC”, “destination port”, “domain name”, “protocol”, “source ip”, “source port” and “tld” additional features were calculated. These features are shown in Table A.3.

Feature	**Description**
Timestamp	The time in seconds since 01-01-1970 on which the alert was raised. This is equal to the UNIX timestamp.
Source ip	The source ip address from which the attacker came.
Source port	The source port the attacker used.
Destination port	The port to which the attacker connected.
Protocol	The protocol the attacker used. This can be for instance tcp, udp or icmp.
Flag	This feature represents the first flag used in honeyd. It indicates the start or end of a connection, and may contain three values: ”S” (indicating the start of a new connection), ”E” (the end of a connection) or ”-” (not belonging to a connection).
Flag2	This flag is more or less used for several different comments, used by honeyd for an alert.
Size	This field represents the packet size the attacker used.

Table A.1: *Direct features used for the experiment*

Feature	**Description**
Domain name	The domain name the attack originated from. This could be “uvt.nl” for example.
TLD	The Top Level Domain the attacker used. In case of the “uvt.nl” domain name, this TLD would be “nl”.
ASN	Autonomous System Numbers (ASNs) are globally unique numbers that are used to identify autonomous systems and which enable an AS to exchange exterior routing information between neighboring ASes. An AS is a connected group of IP networks that adhere to a single and clearly defined routing policy [1]. In case the attacker came from Tilburg University, the AS Number would be “1103”.

BGP	Border Gateway Protocol is an interautonomous system routing protocol. An autonomous system is a network or group of networks under a common administration and with common routing policies (Rekhter, 1995). In case the attack would be originated from Tilburg University, the interautonomous system would be registered as "137.56.0.0/16".
CC	This is the country code of the attacker's ip address, according to the Cymru database. The CC for Tilburg University is "NL".
Registry	There are five regional internet registries Coordination Centers, which provide internet resource allocations, registration services and co-ordination activities that support the operation of the Internet globally. For instance, the RIR of Tilburg University is "ripencc". Other RIRs are AfriNIC, APNIC, ARIN and LACNIC.
Allocated timestamp	The time in seconds since 01-01-1970 that the netblocks have been allocated or reserved for the netblock of the attacker's ip address.
Day of the week	The day of the week the attack was.

Table A.2: *Indirect features used for the experiment*

Feature	**Description**
Timediff1-10	This feature is used to indicate how many seconds ago this value for the direct or indirect feature was seen for the last 1-10 times.
Bool Incident	This feature indicates whether or not the last time this value for this feature was seen, the alert was an incident. It can be either "0" or "1".
Bool Last Alert	Feature incidating whether the last alert was with the same feature value the attacker is using. The value of this feature can be either "0" or "1".
Count Incident	This feature indicates how many times the feature value of the attacker has been reported to be an incident.
Count Seen	Feature representing the number of times this feature value has been seen as an alert.
Count Seen 5min	This feature is used to indicate how many times this feature value has been seen during the last 5 minute time frame. A time frame could be for instance from 15h00 to 15h05.
Count Seen hour	This feature is used to indicate how many times this feature value has been seen during the last hour, using 5 minute time frams.
Count Seen day	This feature is used to indicate how many times this feature value has been seen during the last 24 hours, using 5 minute time frames.
Count Seen week	This feature is used to indicate how many times this feature value has been seen during the last week, using 5 minute time frames.
Count Seen month	This feature is used to indicate how many times this feature value has been seen during the last month, using 5 minute time frames.

Count Seen year	This feature is used to indicate how many times this feature value has been seen during the last year, using 5 minute time frames.
Count Seen plus 5min	This feature is used to indicate how many times this feature value will be seen during the next five minutes.
Count Seen plus hour	This feature is used to indicate how many times this feature value will be seen during the next hour using five minute time frames.
Count Seen plus day	This feature is used to indicate how many times this feature value will be seen during the next day using five minute time frames.

Table A.3: *Calculated features used for the experiment*

Appendix B

Scripts

The scripts that were used for the research discussed in this thesis are available at

Appendix C

Results

The results of the experiments are made available on