Master Thesis

Threat Detection Systems

Author:
Frank van Vliet
s0002720
frank@root66.org

Graduation committee:
dr. D. Bolzoni
prof. dr. P.H. Hartel
dr. F. Kargl

Distributed and Embedded Security Group,
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

October 17, 2010

Contents

Contents **i**

Abstract **1**

1 Introduction **3**
1.1 Scenario 1: risk management 5
1.2 Scenario 2: network security monitoring 6
1.3 Problem definition 7
1.4 Overview 8

2 Risk management **9**
2.1 Standards 11
2.2 Literature 14
2.3 TDS in Risk Management 16

3 Network Security **19**
3.1 Standards 19
3.2 Literature 20
3.3 TDS in Network Security 22

4 Threat Detection System **25**
4.1 Architecture 26
4.2 Intrusion Detection System 28
4.3 Threat classifier 34
4.4 Threat profiler 42

5 Benchmark and results **51**
5.1 Testing methodology 52
5.2 Real network experiment 53
5.3 Hacker experiment 60
5.4 Discussion 68

6 Conclusion **71**
6.1 Recommendations for further research 72

A Attack and Penetration test documentation **77**
A.1 Description 77
A.2 Result 77
A.3 Vulnerability Score 78
A.4 Solution 80

B Snort configuration **81**

C Checklist comparison **85**

D First version of the signatures for the Basic Web Application Scan Checklist **89**

E First version of the signatures for the Advanced Web Application Scan Checklist **93**

F Final version of the signatures for the basic web application scan checklist **97**

G Final version of the signatures for the advanced web application scan checklist **101**

General References **105**

Web References **109**

Abstract

Traditionally, Intrusion Detection Systems generate a large stream of alerts for all the attacks on a network. This thesis presents a Threat Detection System, which generates a list of threats (either human beings or automated computer programs) attacking the network and assigns properties to them such as skill, the intensity of their attacks and whether they are a computer or a human. This allows security experts to focus only on the threats that are dangerous and is the basis for a quantitative approach to risk management.

Chapter 1

Introduction

> "If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology" —*Bruce Schneier*

Traditional Intrusion Detection Systems focus on detecting attack instances on a computer network. The work presented in this thesis is different, since the attack instances are used to construct a profile of the actual attacker. An attacker is named a "threat" and can be a human being or an autonomous computer program. The system is therefore named a Threat Detection System (TDS) and will improve the security of a computer network.

Security is often defined as a combination of confidentiality, integrity, and availability of assets [36]. In the old world of paper and snail-mail, security meant that assets (for example some piece of information) must be kept secret, we must be able to trust the integrity of that piece of information, and the asset should be available to legitimate users when they need it. In the new world based largely on the Internet, security is still important for the same reasons.

When computer systems are connected to the Internet, the systems themselves and the information they contain are assets and their security are threatened by hackers, worms, virii and botnets which are named threats [2], [36]. These threats, which are persons or computer programs, search for weaknesses (named vulnerabilities) of the computers and information (named assets) and try to exploit them using attacks, compromising the assets' confidentiality, integrity, availability, or a combination of those [2], [36]. The definitions are illustrated in Figure 1.1.

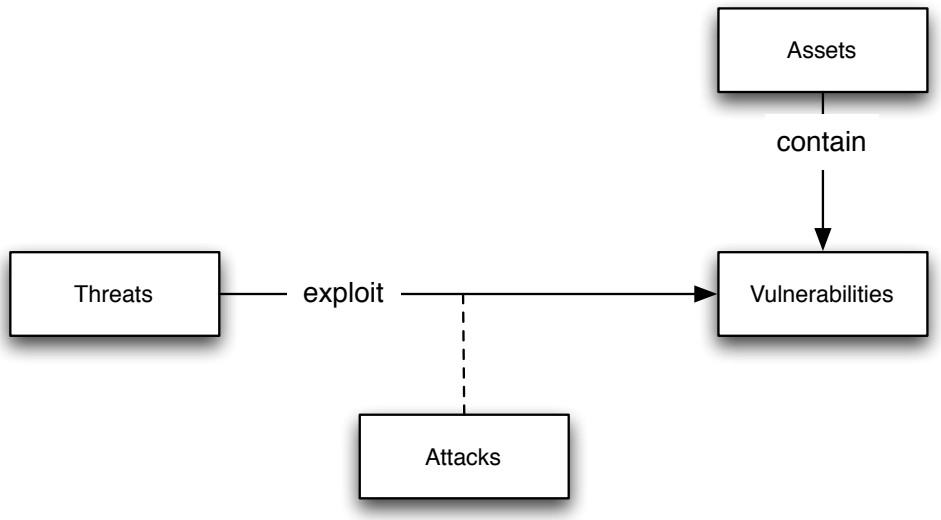

Figure 1.1: Definition of threat, exploit, attack, vulnerability and asset

A computer system is secure when threats cannot exploit the vulnerabilities of that system. There are two approaches towards secure computer systems [2]: The first approach consists of removing the vulnerabilities of the system or adding defenses that makes the exploitation of them harder. The time and skill required for a threat to compromise the system increases, and the threat will (hopefully) stop attacking voluntarily. It is impossible to locate and eliminate all vulnerabilities of a system, therefore using only this first approach is not the best way to secure the system [2]. The second approach consists of the detection of threats while they are attacking the system and stopping them before they are successful in compromising it, for example by removing their access to the system and involving the police. A combination of the two approaches is best, since reacting to threats takes time that can be obtained by eliminating the vulnerabilities [2].

The motivation for this master thesis can be illustrated with two scenarios where a security expert is implementing the two described approaches to computer security and is having problems performing her task effectively. These problems, and the solution to these problems, are the basis for the problem description of this thesis. The first scenario features a security expert who helps a client by identifying vulnerabilities of their network, so they can be eliminated. In the second scenario, the security expert is tasked with securing a network by monitoring all network communications to find threats. The expert faces different problems in each scenario, but can benefit from the same solution that will be described in section 1.3.

1.1 Scenario 1: risk management

In the first scenario, a security expert assesses the security of a website (a web shop selling products) for a client by performing an attack and penetration test. This test consists of both automated and manual testing and results in a list of vulnerabilities which require the attention of the client.

For this scenario, only one vulnerability is discussed: a SQL injection vulnerability. The vulnerability allows threats to manipulate the queries which are executed on the database of the website, enabling the threat to read all the information in the database. Since the database contains all client and order information, this attack would compromise the confidentiality of this information. The full documentation of this vulnerability, as reported by the security company Pine Digital Security, is included in Appendix A.

Each vulnerability poses a risk to the client, who has to decide whether to accept the risk or mitigate it by solving the problem, implementing controls or using insurance. This is the essence of risk management [31]. In the world of commerce, such decisions are ideally based on money; if the risk is expressed in an expected loss dollar-figure, it can be compared to the cost of solving the problem and other mitigation solutions. To calculate the expected loss, the likelihood (what is the chance of the risk happening) is multiplied by the impact (what is the cost if the risk happens) [31]. However, both the likelihood and the impact are hard to quantify.

Determining the impact of the risk (a dollar amount of loss) is a question for the financing department of the client and should not be the responsibility of the security expert. The likelihood however, is something that is asked of a security expert to calculate. Currently, calculating the likelihood is not possible as there are no clear data or metrics on this [31]. Hence, the likelihood is often defined using categories of "high", "medium" and "low" [34], resulting in an approximation of each risk. In this scenario, the likelihood is categorized as "high", since the commonly used NIST standard defines this as: "The threat-source is highly motivated and sufficiently capable, and controls to prevent the vulnerability from being exercised are ineffective" [34]. This definition is inaccurate, since it is not taking into account any quantitative information, for example, whether threats are trying to compromise the website at all.

The problem of selecting a likelihood for each vulnerability can be solved using a divide and conquer approach; the problem can be split into two smaller problems which can be solved separately. The answers of the smaller problems can then be combined to give a quantitative basis to an estimation on the likelihood of a vulnerability. First, the security expert creates a threat profile for each vulnerability to specify the type of

threat that is able to exploit this vulnerability. This profile could include properties like the skill level, the intensity of the attacks, and so forth. Security experts can have a consensus on how these threat profiles are created for each vulnerability, so this is not guesswork. In this scenario, the threat-profile associated with the SQL injection vulnerability could contain the information as presented in Table 1.1.

Table 1.1: Threat-profile for the SQL Injection vulnerability

Attribute	Value	Motivation
skill	basic	SQL injection is a basic skill
android	human	Several manual steps (i.e. logging in) are required to exploit this vulnerability
intensity	any	This vulnerability can be exploited in a single attack

Second, a history of threats active on the system is referenced to ascertain how often a threat capable of exploiting the vulnerability is active on the network. In this example, it could be determined that there are (for example) five threats in the past week capable of exploiting this vulnerability. This does not provide an exact likelihood of the risk, but provides a quantitative basis for companies to prioritize risks that go beyond the "high", "medium" and "low" categories used by current standards.

1.2 Scenario 2: network security monitoring

In the second scenario, the security expert is tasked with monitoring a network of computers for threats. The security expert has two options. First, she could try to read all traffic passing through the network and act when she identifies packets that belong to an attack. It is infeasible to read all packets on any active network. It is possible to sample the packets into flows or streams and use that information to identify suspicious behaviors [32], but this does not provide the same level of details for analysis. The second option of the security expert is to automate the reading of all network traffic using an Intrusion Detection System (IDS). These systems (try to) automatically detect intrusions and inform the security expert about them.

In both cases, the expert is wasting much of her time looking at information that is probably not interesting. Many attacks on the network consist of automated (worm) attacks and script kiddie attacks using a downloadable

tool to exploit well-known security problems [56]. These attacks have no effect on the security of a network when security patches are timely installed on each system. A large team of analysts is required to evaluate all attacks on a network.

To help the security expert focus only on the important information, it would be helpful if the information were somehow classified. The expert would then start by looking at the most important events, zoom in on them and review the events that are related. She would spend no time researching uninteresting worm-attacks and could focus on the few real attacks the network is facing.

To make a better distinction between important attacks and uninteresting ones, one should not only classify the attack, but also the threat performing the attack. There are several types of threats, ranging from professional hackers with many resources to hobbyists who are hacking in their spare time to computer worms without any intelligence. If the network is patched and most of the vulnerabilities are eliminated, the security expert could create a profile of threats that would still be able to compromise the security of the network. If a system would allow the security expert to focus on the threats that match such a threat profile, the expert can monitor the network effectively.

1.3 Problem definition

Both scenarios have illustrated the need for information about threats on a network and their profiles. This master thesis will therefore work on the following problem definition:

> *Design a system to detect and profile threats on a computer network to improve risk management and network security monitoring.*

The problem definition contains the following sub questions:

1. How can a system detecting and profiling threats improve risk management?
2. How can a system detecting and profiling threats improve network security monitoring?
3. How to design a system that is capable of detecting threats on a computer network?

4. How to design a system that is capable of profiling threats?

The methodology for designing such a system consists of the following steps:

1. Research risk management to verify if the likelihood prediction of a vulnerability is a problem and if current research is capable of solving this problem.

2. Research network security monitoring to verify if network security monitoring can be improved by adding the concepts of threats and threat profiles and if current research is already presenting solutions for this.

3. Design the architecture of a system that is capable of solving the problems as defined in the previous two steps. The architecture will consist of at least two components, one for the detection of threats and one for profiling them.

4. Implement a system that meets the problem definition using the defined architecture. The choices made in the design of the system must be documented and where possible, are configurable depending on the specific needs of a security expert performing the risk management or network security monitoring. The system will include an example set of algorithms to profile specific properties of threats. These algorithms are accompanied with guidance on how to interpret the results of these algorithms and how security experts can use them for risk assessment and network security monitoring.

5. Test the system in a real network with real threats following a predefined testing methodology. This test verifies if risk management and network security monitoring benefit from the system.

1.4 Overview

This thesis begins with a discussion on risk management (chapter 2) and Network Security (chapter 3), providing the context of the presented research. Then, the architecture (section 4.1) and implementation (chapter 4) of the system are presented. These chapters describe how and why the system is created. To verify the system works and the problem statement can be answered affirmatively, the system is tested in chapter 5 including the test methodology and the actual results of the tests. This is followed by a conclusion and recommendations for future research in chapter 6.

Chapter 2

Risk management

This chapter first describes the general methods for risk management that could be used for computer security, followed by a description of a standard for risk management that is used in the professional world. These descriptions provide a context and identify problems in risk management as it is currently used. Then, the current state of the art in academic research on these problems is described, followed by an answer to the question "How can a system improve risk management?".

The most basic form of risk management can be derived from the ruling of Judge Learning Hand:

> *Since there are occasions when every vessel will break from her moorings, and since, if she does, she becomes a menace to those about her; the owner's duty, as in other similar situations, to provide against resulting injuries is a function of three variables: (1) The probability that she will break away; (2) the gravity of the resulting injury, if she does; (3) the burden of adequate precautions. Possibly it serves to bring this notion into relief to state it in algebraic terms: if the probability be called P; the injury, L; and the burden, B; liability depends upon whether B is less than L multiplied by P: i.e., whether* $B < PL$*.[22]*

In this basic form, the estimated loss is calculated and compared to the cost of solving the problem. When the solution is cheaper than the expected loss, the solution is implemented, else the risk is accepted.

The tradition methods for risk management used a similar formula to calculate the Annualized Loss Expectancy (ALE) (Equation 2.1). This formula considers the impact of each harmful outcome ($\{I_1, ..., I_n\}$) and the frequency each of these outcomes occur ($F_1, ..., F_n$) [30]. One of the problems

of this method is that it suffers from a lack of actual data to estimate values for impact and likelihood [31]. In the mid-1980s, a framework for computer security risk management emerged from this model. To calculate the risk, each combination of the assets, security concerns, threats, and vulnerabilities must be evaluated. This is a task of infeasible proportions [31].

$$ALE = \sum_{i=1}^{n} (I_i F_i) \tag{2.1}$$

Based on this Annualized Loss Expectancy (ALE) method of risk management, different methods were developed to manage risks [26]. The first method is the Integrated Business Risk Management model. This model allows security risks to be managed like other business risks [31]. These risks are identified and then managed using financial (contracts and insurance) or strategic (avoidance, control, cooperation, imitation, and flexibility) [26] management. A problem with this method is that it simplifies the security risks, because they are handled the same way as other business risks and therefore focus on bottom-line business impact instead of the computer security interactions [31].

The second method consists of the valuation-driven methodologies. These methodologies simply ignore the likelihood and only focus on the value of each asset. This greatly simplifies the risk management, but results in either over-securing or under-securing since these methods have no notion of the likelihood of each problem [31].

The third method consists of the scenario analysis approaches. These approaches consist of the analysis of possible scenario's by which computer security is compromised [31]. An example of this is the execution of an attack and penetration test, where the system is compromised on purpose to detect the vulnerabilities and their associated scenario's. Since there is no guarantee that all scenario's are found, this leads to a false sense of security [31].

The final method consists of using best practices. These practices are ofter the industry standard of doing things as well as possible. They consist of design and implementation requirements that are already tailored to the possible threats and vulnerabilities of this industry. A problem with this method is the uncoupling of best practices from the actual risks involved, where best practices are implemented because they are best practices without regard to the actual risks involved. [31].

Currently, performing a perfectly accurate risk management for computer security is still impossible; the traditional method, based on the Annualized Loss Expectancy, requires an assessment of infeasible proportions and

newer methods that require less work fail to accurately capture all important aspects of risk management [31]. Furthermore, all methods suffer from a lack of actual data to estimate values for impact and likelihood [31]. If the methods could use actual data to estimate impact or likelihood, then risk management could be improved [31]. To identify how the lack of actual data is currently affecting risk management as it is used in the professional world, the following section will describe a commonly used standard for risk management in information technology systems.

2.1 Standards

In the professional world, where risk management is used in real situations, a few standards are used. First, there is the Risk Management Guide for Information Technologies Systems [34] by the National Institute of Standards and Technology. This guide contains a process for risk assessment and for risk management. Second, there is a standard by the International Organization for Standard named "ISO27005: Information security risk management". For this research, the NIST standard is used as it is open and specifically focusses on risks.

In 2002, the National Institute of Standards and Technology (NIST) released the Risk Management Guide for Information Technology Systems [34]. This guide implements risk management in two steps. First, a risk assessment is executed by identifying systems, threats, vulnerabilities, existing controls, likelihood, impact, and finally determining the risk and recommending further controls. Secondly, a risk mitigation process is executed to accept, avoid, limit, plan for, solve or transfer each risk. This method for risk management could be an implementation of the Integrated Business Risk Management model earlier discussed, as the standard states:

> *Because the elimination of all risk is usually impractical or close to impossible, it is the responsibility of senior management and functional and business managers to use the least-cost approach and implement the most suitable controls to decrease mission risk to an acceptable level, with minimal adverse impact on the organization's resources and mission.[34]*

Note that this model uses only qualitative values ("high", "medium" or "low"). Risk assessment by the NIST standard consists of nine steps, which are summarized as following:

Step 1: System Characterization Identify all system-related information. This includes the hardware, software, system interfaces, system

missions, and the persons who support and use it. This also includes the data that are stored within the system and their requirements on availability, integrity, and sensitivity.

Step 2: Threat Identification List the possible threats (named threat-sources in this standard) of the system (hackers, criminals, terrorists, spies, employees). For each threat, the motivation (challenge, destruction, monetary gain, revenge, competitive advantage) and the possible actions (hacking, social engineering, bribing, bombing) are identified.

Step 3: Vulnerability Identification Identify all vulnerabilities in the system. These vulnerabilities can be gathered using automated scans, penetration testing and auditing. Each vulnerability should be matched to the actions of threats.

Step 4: Control Analysis Identify all controls implemented in the system or organization to minimize or eliminate the likelihood or impact of vulnerabilities. Possible controls are access control, intrusion detection systems and encryption.

Step 5: Likelihood Determination Estimate the likelihood of exploitation for each vulnerability. This is based on the motivation and capability of the threats and the existence and effectiveness of the implemented controls. The likelihood is a qualitative value with three options:

- High, the threat is highly motivated and sufficiently capable and controls to prevent the exploitation of this vulnerability do not exist or are ineffective.
- Medium, the threat is motivated and capable, but controls are implemented to hinder successful exploitation of the vulnerability.
- Low, the threat lacks motivation or capability, or controls are implemented to prevent or significantly hinder the vulnerability from being exploited.

Step 6: Impact Analysis Estimate the impact of successful exploitation for each vulnerability. Like the likelihood determination, the value for impact analysis is also qualitative with three options:

- High, exploitation of this vulnerability results in high costs or significantly violates, harms, or hinders an organization's missions, reputation or interest.

- Medium, exploitation of this vulnerability leads to considerable costs or violates, harms, or hinders the organization's missions, reputation or interest. A medium value is also attributed to vulnerabilities whose exploitation leads to human injury.

- Low, the exploitation of this vulnerability leads to some costs or noticeably affects the organization's mission, reputation or interest.

Step 7: Risk Determination Determine the risk for each vulnerability. The risk is calculated based on the product of the likelihood value (high = 1.0, medium = 0.5, low = 0.1) and the impact value (high = 100, medium = 50, low = 10) = Risk. The product of these values is then transformed to a qualitative value using the following scale: 1<= low <= 10; 10 < medium <= 50; 50 < high <= 100. The inaccurate and arbitrary definition of risk as defined is the step is the motivation for this thesis.

Step 8: Control Recommendations List possible controls and alternative solutions. During the risk assessment process, the possible solutions and controls are already introduced. For example, a vulnerability in a web site could be solved by changing the code or solved by implementing an application level firewall to prevent this vulnerability from being exploited. Each proposed control or alternative solution should consider their effectiveness (is it solving the problem), legislation and regulation, organizational policy, operational impact and safety and reliability.

Step 9: Results Documentation Document the results. This documentation could describe the risk assessment approach (participants, techniques used to gather information and a description of the qualitative values used), the system characterization (output of Step 1), the threat statement (output of Step 2) and finally a list of vulnerabilities and their gathered information.

One problem of the risk assessment is the lack of actual data to estimate the likelihood and impact [31], as described in the previous section. For the likelihood, this standard uses a qualitative value ("high", "medium" or "low") is based on abstract concept such as motivation and capability of threats.

To illustrate this problem, an example is provided using a web-shop that is vulnerable to SQL injection. To calculate the risk of this vulnerability, step one has identified that the customer information within the database has requirements on confidentiality, integrity, and availability. Step two

has identified hackers on the Internet as a possible threat which can perform hacking actions (those qualifications are directly copied from [34]). To perform step three, an attack and penetration test is executed by Pine Digital Security [58], a Dutch company that specializes in security testing. The documentation about the SQL Injection vulnerability is included in Appendix A. Then, step four, identifies controls that can minimize or eliminate the likelihood or impact of this vulnerability, which in this example are not existent. The fifth step is the determination of the likelihood. Based on this standard, a value of "high" must be given since the threat is motivated and capable, and no controls are implemented to prevent the threat from exploiting the SQL Injection vulnerability. Then, step six, estimates the impact of this vulnerability to be "high", because the confidentiality of the customer database including the credit card information can be compromised completely. If this information is published online, it would significantly harm the organizations mission to sell products online. The likelihood and impact are multiplied in step seven to determine the risk of this vulnerability to be "high". The documentation included in Appendix A also describes the recommended solution that is part of step eight. Finally, in step nine, the previous steps are documented and the vulnerability is reported as being a "high" risk.

In this example, steps two (threat identification), five (likelihood estimation), six (impact estimation) and seven (risk determination) are vague and fail to accurately describe the vulnerability. This problem was already described in the previous section and is attributed to a lack of actual data [31].

If a system is designed to provide historical information on the number of active threats on a computer network, the threat identification, and likelihood estimation could be improved. This system will not solve the impact estimation or risk determination problems, and will not be able to exactly state what the chance is for a vulnerability to be exploited. But, it is possible to define a profile based on motivation and capabilities and know how many threats met this profile during a specific time, thereby improving the likelihood estimation.

2.2 Literature

In the academic world, the lack of quantitative data is a known problem and different systems are proposed to obtain this. However, these systems have their own set of problems, since their applicability to the real world is often limited.

In 2004, Gehani and Kedem proposed the first system to automatically

calculate the risk of a system and respond accordingly [13]. This system calculates the risk based on events from an Intrusion Detection System and manages this risk by altering the exposure of the system; each request for access is granted based on the current risk level of that system. Although the system is designed to autonomously respond to risks, which is infeasible in real world setting, the system does automatically calculate the likelihood component of each risk in a quantitative manner. However, the likelihood is calculated by testing whether the attack matches completely or only partial to some specification. This could improve the likelihood calculation of known attacks, but not for new vulnerabilities for which the attacks cannot be enumerated.

In 2008, He et al proposed a system using game theory [16]. This system also identifies the lack of qualitative data and computes this using a game theoretical model. According to this model, the likelihood component of a risk is based on the cost and benefits for both attackers and defenders. An equilibrium is calculated and assumed to be the likelihood component. However, two major assumptions are made by this model, which are not in line with real world situations. First, attackers are assumed to be equal; there is no difference in likelihood for one threat to another. This is obviously not in line with the real world, since some vulnerabilities can only be exploited by attackers with a certain level of skill. Second, an attack will always succeed when no defensive measurements are implemented. This is not in line with the real world since the success depends on motivation, skill set and environmental elements such as authentication.

In 2009, Li and Gou proposed a method using a Hidden Markov Model (HMM) [23]. Although the usage of HMM has been proposed prior to this research [15], the research of Li and Gou uses a real-time quantitative HMM network to continuously update the risk values for each host. By using the alerts from an IDS, the risk is calculated based on the severity, the key level of the asset, the priority, and the reliability. This is not according to the likelihood and impact values of the traditional risk assessment methodology, but provides a score from 0 to 9 that is an improvement over the "high", "medium" and "low" qualification values used in this traditional methodology. The main problem of this method however, is that research focusses only on known vulnerabilities in commonly used software. Since these vulnerabilities are mitigated by implementing a proper version management process, the research fails to solve the problem of real world risk assessment.

2.3 TDS in Risk Management

Currently, Risk management lacks adequate data to quantitatively determine the likelihood of a risk and therefore improve from having actual data on the threats on a network and their profiles. When designing the Threat Detection System (TDS), it is important to ensure that it can provide this information on a real network.

For each vulnerability, the security expert assessing it must be able to select applicable threat profiles; the types of threats that are able to exploit the vulnerability. The system should then return the number of threats meeting these profiles that were detected during a specified time. If, for example, the risk of a vulnerability is assessed that requires advanced skills, for example, an XML External Entity Attack, the system should only return the number of threats that have the capability of performing such advanced attacks. Instead of using vague concepts such as motivation and capability, the exact number of threats capable of exploiting the vulnerability is presented.

The profile of a threat consists of several properties. The security expert should define the threat profiles by selecting the properties that are required to exploit the vulnerability. For this research, the properties to estimate the likelihood value as defined in the NIST standard [34] are chosen. However, because other security experts can disagree on the properties they would like to use; the system must be designed to accept properties that are defined and programmed by other security experts.

The following list describes the properties that are selected for this research. These provide the security expert with a three properties, and implementing them illustrates the working of the TDS.

Skill class A threat has several skills that are used to exploit vulnerabilities (the NIST standard calls them capabilities). While it is tempting to be able to select the exact skill that is required to exploit a vulnerability, this assumes an ability of the TDS to detect all possible skills of a threat. Because the system will be used on real networks, it will be impossible to configure the TDS to detect all possible skills. When using skill classes where threats possessing one skill are likely to possess the other skills in the same class, the TDS can be configured to detect a portion of the skills within the class to still provide useful data. This is a tradeoff between completeness and accuracy and must be configurable in the TDS to be changed by the security expert executing the risk assessment.

Android class Some vulnerabilities require the threat to be an actual human to exploit it, for example, when the threat first must register

and then log in with her new credentials. This is not possible for automated tools, scanning the Internet for vulnerabilities to exploit.

Intensity class The intensity of the attacks by a threat can be relevant information as it is closely related to motivation. There is a difference in threats that try some attacks for a few minutes and threats that actively spend months attacking a system or network.

It is important to note that for all three properties, the TDS must consider a possible hierarchy of classes. A threat performing a skill in an advanced skill class will also possess the skills in the basic skill class. Likewise, a threat capable of performing manual attacks is also able to download a tool and perform automated attacks.

When the likelihood of a vulnerability is estimated, the TDS can be queried for the number of threats possessing a specified skill and intensity class over a specified time which can improve the likelihood estimation and thereby the risk assessment of a vulnerability.

Chapter 3

Network Security

The security of a computer network is based on three components: confidentiality, integrity, and availability. These components apply to the assets (data or systems) within the network [36]. To protect those assets, there are two approaches that are commonly implemented [2].

The first approach to network security is analog to building a large wall around a city; the bigger and higher the wall is; the harder it is for threats to get inside the city. It is universally accepted that there is no such thing as perfect security, which means that a threat with sufficient capabilities and resources is always able to climb over the wall.

The second approach to network security consists of monitoring the network for threats. Using firewalls, those threats can be stopped before they compromise the network.

The best network security is achieved using a combination of the two, making it hard for a threat to compromise the network and allowing the network security staff to stop them before they succeed. This combination could be modeled in Risk Management practices by defining the process of stopping the threats as controls that decrease the likelihood a threat can exploit a vulnerability [34].

3.1 Standards

The professional world of computer security focusses mainly on the first approach to network security: making it as hard as possible for a threat to compromise the network. This includes installing the latest patches, installing firewalls, and ensuring that there are no vulnerabilities in the software (a detailed list of security measurements to be taken is documented

in the Certified Secure Checklists [61], [62], [63] and [64], and in various system security handbooks [54], [48]). Often, an attack and penetration test is performed to reveal vulnerabilities which are subsequently resolved (possible using a risk management approach as described in the previous chapter). The security that is achieved by this approach is defined by the motivation of the threats; the network is secure if threats stop their attacks voluntarily: because they get bored, it takes too much effort, or another network is easier to compromise and therefore is more attractive. This means that the security of a network depends on the threats stopping their actions.

The second approach to network security is the active stopping of threats. Using some form of detection (either manual, automated or a combination), threats that attack the network can be detected and preventative measures can be taken to stop them. Richard Bejtlich, author of the book "The Tao of Network Security Monitoring: Beyond Intrusion Detection" [2], is one of the leading experts on network security monitoring in the professional world. His ideas on network security are invaluable for security professionals working with real networks that have to be defended against real threats. According to Bejtlich, an important part of building a defensible network (next to limiting the intruder's freedom, offering only a minimum number of services, and ensuring the network can be kept current) is the ability to watch the network. This should be anticipated during the design of a network and allows for the ability to audit all critical parts of the network. Watching a network results in indications and warnings. Indications are actions that are observable or discernible and confirm or deny enemy capabilities and intentions [2]. These indications are what Intrusion Detection Systems call alerts. These indications are interpreted by analysts who generate warnings when indications are serious. It is important to note that the collection of information can be automated ([59], [42], [50], [47]), but the analysis always requires real people [2]. The escalation of an indication to a warning should be decided by a human, capable of analyzing the context of the indication.

3.2 Literature

Many different types of Intrusion Detection Systems are developed to aid network security monitoring. These systems automatically detect attacks and report them in so called alerts either directly to the security experts tasked with monitoring the network, or via a Security Information Management server that aggregates the alerts (illustrated in Figure 3.1).

There are different distinctions in Intrusion Detection Systems [1]. First,

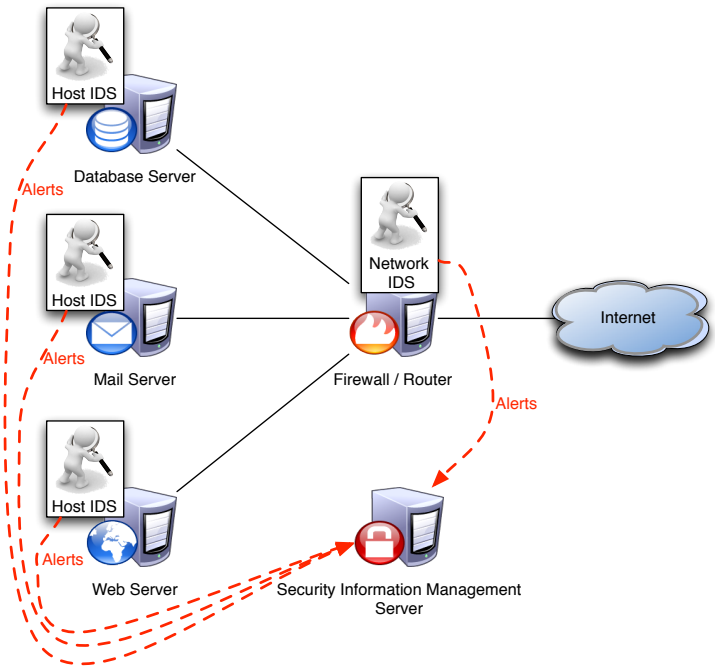

Figure 3.1: An example deployment of IDS in combination with SIM

there are host based and network based systems; the Host based Intrusion Detection Systems are installed on the computer they are monitoring, analyzing the log files and changes to the system and Network based Intrusion Detection Systems are installed on a network and analyze the network traffic for attacks. Second, there are two approaches in how attacks are detected: anomaly based and signature based.

In Anomaly based Intrusion Detection Systems [21], [24], [38], [40], a profile of supposed good traffic is created and new network traffic is matched to this profile of good traffic to find anomalies. This system requires a training phase to create the profile of good traffic. This is a problem since the security expert must train the system with known-good network traffic, which is infeasible in real networks; the expert would need to capture weeks of network traffic and then manually remove all attacks to create this known-good traffic and when the network changes, for example by adding a server, the known-good traffic is no longer representative for the network and has to be created again. Also, anomaly-based IDS cannot distinguish between the different type of attacks; they can only state that the traffic is different from normal traffic [17]. These problems are the topic of ongoing research [6], [4], [5], [7], [32].

The other approach to Intrusion Detection Systems is signature based [1].

This approach (similar to anti-virus) employs a set of attack signatures and matches them against network traffic. All traffic matching a signature is considered to be an attack and is reported as such. This type of IDS can only detect attacks for which a signature is created, which means that it cannot detect new attacks [1].

A problem of IDS generally is that it generates a large stream of alerts that is too much to process manually. A solution to this problem is the use of Security Information Management (SIM) systems, which combine the results of multiple IDSs [12, 51]. Such systems allow the security expert to view the alerts of multiple Intrusion Detection Systems (as illustrated in Figure 3.1). Using SIM, several techniques can be implemented to reduce the number of alerts and therefore the workload of the security experts monitoring the network [29].

The first type of reduction in number of alerts consists of correlating alerts that are similar using different data mining techniques and distance functions [8], [20], [37], [33]. When multiple Intrusion Detection Systems are used, one attack can trigger multiple alerts (one from each IDS) and using these correlation systems, these alerts are clustered in one high-level alert describing the complete attack. Also, with many worms and bot-nets active on the Internet, many identical attacks are launched from many different systems. These correlation systems can cluster these alerts of these attacks in one high-level alert containing all those identical attacks.

The second type of reduction in number of alerts consists of using attack scenarios, also called attack plans [3], [9], [10], [35]. Threats perform attacks based on an attack plan, which consists of different states until some target is reached [3]. This reduction combines multiple attacks in one alert describing the complete sequence of attacks.

The third reduction in number of alerts is achieved by using attack trees [27], [28]. By creating an attack tree containing the attacks and how the correlate, it is possible to see whether high-level goals are reached.

3.3 TDS in Network Security

The staff monitoring the network for threats could greatly benefit from a system that can automatically detect problems. Current systems (Snort [59], Bro [42], ModSecurity [47] and OSSEC [50]) are only capable of detecting known attacks which lead to a long list of attacks that have no real impact on the network. The staff needs to go through an endless list of attacks which don't even contain the interesting events.

The TDS could help the monitoring staff by presenting a list of threats and their skill and intensity(as described in TDS in Risk Management (section 2.3)). Since most threats have no chance of compromising the network (because they lack the required skill class or intention class), the staff can focus only on threats that have an actual chance of compromising the network.

Security Information Management systems allow the attacks to be correlated, but currently do not assign properties to threats. This research could be used to as part of these systems.

Chapter 4

Threat Detection System

This chapter describes the architecture and implementation of the Threat Detection System (TDS). The requirements for the system are defined in the previous two chapters and are the following:

1. Detect threats on a real computer network.
2. Determine the profile of these threats using threat properties that can be configured or defined by security experts.
3. Implement a skill, android, and intensity property as examples.
4. Allow a security expert to specify a threat profile and timeframe and return the matching threats.

Based on these requirements for the prototype TDS, the following decisions are made:

1. The TDS is plug-and-play. There is no time, and money required to adapt the system for a specific network. This allows the system to be used directly in a new network.
2. The TDS connects passively to the network, for example using a SPAN port on a switch or router. Therefore, the system does not influence the integrity and availability of the network in any way. An alternative would be the installation of sensors or each computer (to allow host based intrusion detection), but then the system will not be plug-and-play.
3. The TDS supports IPv4. In the future, it would be useful to include IPv6, but this is beyond the scope of this prototype.

4. The TDS supports HTTP. Nowadays, most attacks (60% of the attacks observed on the Internet [44]) are performed on websites and limiting the scope of the prototype to this protocol demonstrates the working of the TDS without the implementation of every possible protocol.

First, the architecture of a system that is capable of implementing these requirements is described in section 4.1, followed by the implementation of the separate components in section 4.2 to section 4.4. For each component, the choices that are made are documented. section 4.4 also provides guidance for security experts on using the implemented threat properties.

4.1 Architecture

This section describes the architecture of the Threat Detection System (TDS). It describes the ideas behind the system and describes the function of each component. The actual implementation of these components can be replaced to adapt the system to different environments.

The most important concept behind the Threat Detection System is the shift from detecting attacks to detecting the actual attackers (threats) and their properties (such as skill). Detecting threats is accomplished using the output of one or more Intrusion Detection Systems (each detecting attacks) and clustering the attacks in groups of attacks that belong to a single threat. The result of this process is a list of threats and their attacks. The Intrusion Detection Systems are not required to detect 100% of the attacks, since one attack for each threat is theoretically sufficient to detect that threat. Then, for each threat, a set of properties like skill is determined based on the attacks they performed or some external knowledge database. The accuracy of the properties determined for each threat depends on the amount of attacks detected of this threat; more detected attacks of a threat increase the accuracy of the properties determined for this threat.

The TDS therefore consists of three components which are illustrated in Figure 4.1: one or more Intrusion Detection Systems (IDS), a threat classifier and one or more threat profilers. This architecture allows one component, for example, the implementation of the IDS, to be replaced by a different IDS or use multiple IDS concurrently. It is also possible to scale the system by installing many IDS components on separate systems that all report to one thread classifier.

The responsibilities for each component are defined as following:

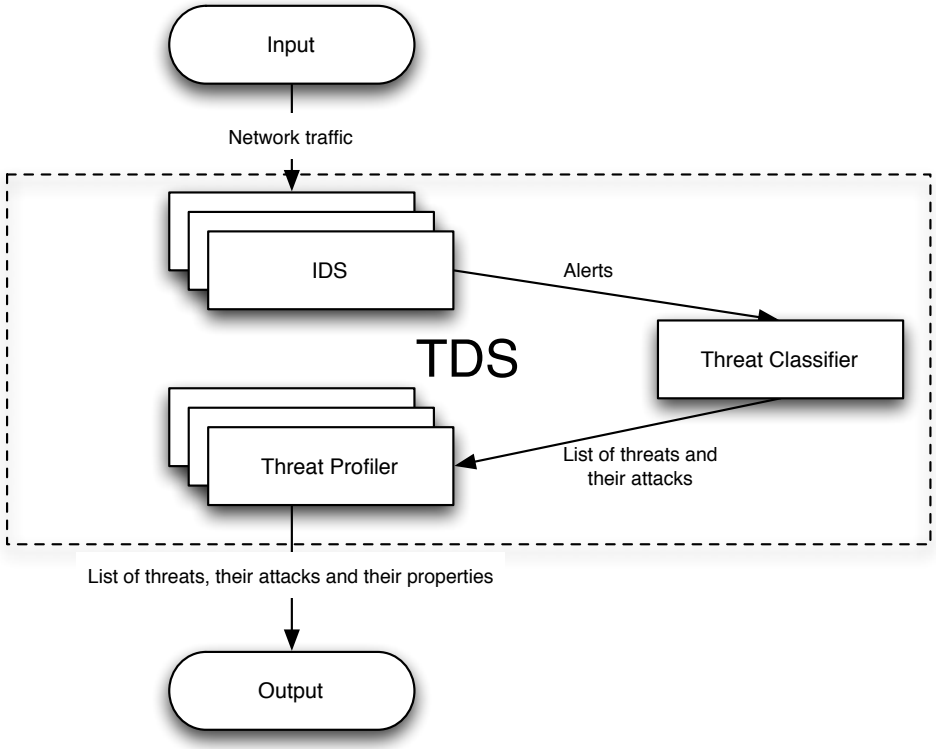

Figure 4.1: Components of the TDS

Intrusion Detection System The Intrusion Detection System (IDS) is responsible for detecting attacks in the network traffic. Different types of IDS could be used, but they must be able to detect actual threats on actual networks. To allow the threat profiler to determine the skill of a threat, the IDS must be able to identify the attack method used for each attack. The architecture of the system does not require an 100% detection rate from the IDS, since a few attacks of each threat already allow the threat to be detected and profiled. When more attacks of the threat are detected, the accuracy of the threat profile increases.

Threat classifier The classifier (a clustering algorithm) translates the alerts generated by the IDS in a list of threats and their attacks. This translation can be performed based on (a combination of) the MAC address, the IP address, or the user name, depending on the network traffic. The architecture does not dictate a specific way of classifying the threats, but the performance of the system increases when the classifier is more accurate in grouping the alerts per threat.

Threat profiler The threat profiler is responsible for determining a specific property of a threat, such as the skill. To perform its function, a threat profiler receives a list of threats and their attacks from the threat classifier component. It is important to note that only concrete properties can be determined. Abstract properties like "intent" and "motivation" seem logical properties to profile, but cannot be reliably assessed and should not be used. Subjective properties that depend on some interpretation, like skill, can be used when they can be reliably assessed given a specific interpretation, but the interpretation must be configurable since other users of the system could disagree with the interpretation and want to change it.

The architecture of the Threat Detection System provides a novel way of intrusion detection: a shift from detecting attacks to detecting threats and their properties. The architecture allows the system to be adapted to different environments and to use third party components when available.

4.2 Intrusion Detection System

The Intrusion Detection System is the component that detects the attacks on a network. Each attack is represented by an alert that is sent to the threat classifier for further processing. This section describes the choices for the used software and configuration of this software. Furthermore, a new set of signatures to detect generic attack types is presented.

Software

There are many Intrusion Detection Systems available such as Snort [59], Bro [42], Modsecurity [47] and OSSEC [50]. A selection was made using the following requirements:

- The IDS must directly work on any network as this is a requirement of the TDS. The system can be placed in any network and should just work, without a training period to adapt to the network. This means that the IDS cannot require training or a special configuration to work on a network and that anomaly-based IDS are not an option.
- The IDS must be able to passively monitor IPv4 network traffic; more specifically, it must be able to monitor the traffic from a live interface. This means that host based IDS and systems that are part of the server software are not an option.

- The IDS must be able to reassemble IP fragments, TCP streams and decode HTTP traffic. Each layer in the OSI model has it's own encoding and fragmentation techniques and those must be handled by the IDS as all signatures in the IDS depend on the unambiguity of the network traffic. This is a vulnerable part of any signature based IDS, since the reassembling and decoding is specific for each operating system, software product, network layout, and temporal information like the ordering of packets and the time between them. All signature based IDS can be evaded by exploiting this weakness, but the IDS for this TDS does not need to be perfect. Unlike a traditional IDS, where every undetected attack is a problem, the TDS can still perform as long as some attacks of the threat are detected, but is less accurate.

- The IDS must use signatures that match the attack method in stead of exact attacks (or exploits). Real threats to networks create their own attacks and are also be able to exploit custom software. This means that signatures only matching known vulnerabilities in specific applications or only match specific exploits are unacceptable.

Currently, none of the third party IDS products available meet all the requirements. Snort [59] and Bro [42] can satisfy all except one requirement; they fail to use generic attack signatures and therefore only detect known exploits [57]. ModSecurity [47] is a product that is able to detect generic attack signatures, but is host based and does not meet the passive monitor requirement. Since Snort is the generally preferred IDS [46], it is chosen as the third party IDS product to be used in this TDS. A custom set of signatures is created to fulfill the requirement on generic attack method signatures.

Configuration

The configuration of the Snort IDS consists of single configuration file that is included in Appendix B. This configuration is basically instructing Snort to listen traffic on TCP port 80 (HTTP). It will verify the IP and TCP checksum of incoming packets and will handle fragmentation in both IP and TCP. More information about the configuration is included in comments in Appendix B.

Signatures

The signatures accompanying Snort are inadequate to detect real threats performing real attacks; the signatures detect only specific exploits for

specific vulnerabilities in specific software [57]. These signatures work only adequately when protecting against computer worms performing one specific attack on a known vulnerability in a commonly used software product. Since most of the websites consist of custom software, using the signatures accompanying Snort is not useful and a new set of signatures is created for use in this Threat Detection System.

The attacks performed by real threats can be described using generic attack methods, such as SQL Injection and Cross Site Scripting. To develop a set of signatures that match these generic attack methods, a complete overview of these attack methods is required. There are some lists available describing the attack methods and three resources are particularly interesting: the OWASP top 10 [55], the CWE/SANS Top 25 Most Dangerous Software Errors [43] and the Certified Secure Web Application Scan Checklists [64], [62].

The OWASP top 10 is a list of the 10 most critical security risks in web applications [55], grouped in rather generic categories as "Injection Flaws" or "Broken Authentication and Session Management". The CWE/SANS Top 25 Most Dangerous Software Errors [43] is a list of the 25 most dangerous problems found in software in generic. Both lists are not useable since they list only a subset of attack methods.

The Certified Secure Web Applications Scan Checklists list all possible attacks that can be performed by beginning threats (the basic scan checklist [64]) and advanced threats (the advanced scan checklist [62]). Although it is not possible for these checklists to be 100% complete, they provide much better coverage of the attack methods than the OWASP Top 10 and the CWE/SANS Top 25. A comparison between the three is provided in Appendix C, which illustrates the coverage of the basic and advanced checklist when compared to the OWASP Top 10 and the CWE/SANS Top 25.

For many items in the Certified Secure Web Application Scan Checklists, one or more signatures are created with an annotation to the specified checklist item they match. However, for some checklist items, it was not possible to create signatures that would not be triggered accidentally by legitimate users of the systems. An example of this is the checklist item "Check for identifier based authorization" (basic scan checklist item 1.3). The documentation states [60]:

> *Identifier based authentication is using an identifier (the id=5 parameter in a url) to allow access to an object. If you are for example authorized to view the following url:*
>
> - *http://server/userinfo.php?id=23421*

Then you can change the url to something different and view the user information of someone else:

- *http://server/userinfo.php?id=23422*

Creating a signature to match this behavior, would require the IDS to report any changes of a number in the URL. Normal browsing behavior can easily match this behavior, and such a checklist item is therefore not implemented in the signatures.

A first version of the list of signatures for the basic and advanced checklists is included in Appendix D and Appendix E. These appendixes include comments about the checklist items for which no way was found to match only the attack and no legitimate network traffic. The developed set of signatures covers 35% of the attack methods as defined by the checklists. Although this appears to be a low percentage of the known attack methods, they cover most of the common attack methods such as SQL Injection and Cross Site Scripting (ranked 1 and 2 in both the OWASP Top 10 and the CWE/SANS Top 25). In terms of performance, threats could evade this system using only attacks that are not implemented in signatures. However, this means that they cannot use many popular attack methods, significantly hindering their possibilities. Future research should investigate the implementation of more checklist items in the signature set, to provide better coverage, and increase the performance of the system (allowing the system to detect more threats).

Signature tuning

The performance of the Intrusion Detection System component is important for the performance of the complete Threat Detection System. A common criterion of IDS is the false positive rate, indicating the number of detected attacks that are not attacks. In the TDS, a false positive in the IDS can introduce a threat which is not a threat at all, which is a false positive in the TDS.

To test the false positive rate of the signatures, the IDS is placed within a real network hosting around 100 websites for 28 days. Because this test is executed on a real network, the IP addresses, exact URLs, and identifying headers are anonymized in all examples (replaced with the word "anonymized") to preserve privacy. A total of 8115 attacks were detected by the IDS of which 2699 were false positives. This is a false positive rate of 0.33 and is a bad result for any IDS. To understand this high rate of false positives, it is important to closely examine the cause. In Table 4.1, a list is

presented of IDS signature and their true positives and false positives. This table clearly shows that one rule is the cause of most the false positives.

Table 4.1: Preliminary Test: Signatures, True Positives and False Positives

signature	true positives	false positives	checklist item
1120001	3	0	Check for default and predictable accounts
1120002	1	0	Check for default and predictable accounts
1210001	250	2578	Check for filename injection / path traversal
1210002	3	0	Check for filename injection / path traversal
1220001	187	0	Check for SQL injection
1220002	354	0	Check for SQL injection
1220003	1833	0	Check for SQL injection
1230001	176	0	Check for cross site scripting
1230003	13	0	Check for cross site scripting
1230005	4	0	Check for cross site scripting
1240001	5	0	Check for system command injection
1240002	0	1	Check for system command injection
1310001	15	0	Check for uploading of (dynamic) scripts
2091001	0	2	Check for uploading outside of intended directory
20210001	2431	88	Check for extraneous files in document root
20340001	6	0	Check for accessible CVS/SVN directories
20340002	7	0	Check for accessible CVS/SVN directories
20650001	0	30	Check for brute-force user name enumeration
20770001	3	0	Check for SSI injection
20790001	125	0	Check for dynamic scripting injection

Of the 2699 false positives, 2578 false positives are generated by just one signature (1210001), matching path traversal attacks where the threat is allowed op open files and uses ../ to open files different directories on the system. It is possible to just delete this signature matching path traversal. However, the signature also matched 250 true positives which would no longer be detected by the IDS. When the contents of the false positives in reviewed, 2555 of the false positives are created by just one IP address that keeps sending requests as illustrated in Listing 4.1. Another user copy-pasted the URL found on the Google result page. This URL contained three dots since it was abbreviated as illustrated in Listing 4.2. Other false positives for this signature are listed in Listing 4.3.

```
GET /anonymized HTTP/1.1
Accept: text/plain,text/html,*/*;q=0.3
TE: trailers
Host: anonymized
```

```
Referer: ../../
User-Agent: Mozilla/4.0 (eknip/1.60)
Connection: TE
```

Listing 4.1: 1 of the 2555 similar false positives by one IP address. The host was indexing large parts of a website and used ../ in the Referer header field. This field is used to communicate to the web server which page was visited previously to this page and allows websites to know which other websites link to them. Using ../ in this field is highly unlikely, but the host in this case did not try to attack the network.

```
GET /anonymized/.../anonymized.html
```

Listing 4.2: A user copied an abbreviated URL from Google instead of clicking it. The abbreviation introduced the ... in the URL

```
GET /nl/../../../../anonymized HTTP/1.1
GET /anonymized/../anonymized.gif HTTP/1.1
```

Listing 4.3: Two other false positives for signature 1210001, matching path injection. In this case, the browser is erroneously sending the ../ content, thereby triggering the signature

The false positives for the signature 1240002 were triggered by a user uploading a video file that happened to contain the signature for system command injection. Another false positive was for signature id 2091001, where a user uploaded a file with the name "webcam-t01 kleiner..jpg" which triggered the signature that looks for uploading outside the directory (it matched ".."). This signature is improved to match "../" and ".." instead of just "..".

The false positives generated for signature id 20210001 were matched because users were legitimately visiting sites containing /admin, /website and /test. The signature matches when a user scans for extraneous files in the document root and triggers when more than 10 such requests are made within 60 seconds. The three hosts that triggering the true positives were each using more than 100 requests within 60 seconds. Therefore, signature will be improved by raising the limit to trigger this signature to 100 requests within 60 seconds.

Finally, the false positive generated for signature id 20650001 was matched because a user forgot his or her password 30 times. Here, the signature will not be changed since this warrants further investigation.

In the end, manually handling 2699 false positives in 28 days averages to around 96.4 false positives a day. This is impractical to work within real live situations. To improve the rate, the signatures are improved (Appendix F, Appendix G.

4.3 Threat classifier

The threat classifier receives a list of attacks on the network and groups them based on the threat. This results in a list of all threats and for each threat a list of their attacks.

There are many different ways of classifying a set of attacks, but this problem has two special properties. First, there is no training set available as part of the requirements of the system. Without a training set, the algorithm must use unsupervised classification. Second, the result should be a grouping of the attacks instead of a model that can classify new attacks in the correct group. Based on these properties, the classification requires a clustering algorithm:

> Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters)[18].

Other classification algorithms either try to detect if a pattern is part of a group (for example Bayesian classification or minimum error rate classification) or the algorithms require a training set (discriminant analysis) [11]. Such algorithms cannot be used as classifier in this research.

The clustering algorithm typically consists of the following steps [19]:

1. pattern representation, describing the pattern (attack in this case) into features,
2. definition of a distance function between to patterns,
3. clustering or grouping,
4. data abstraction (if needed), and
5. assessment of output (if needed).

The following sections will each discuss one step of the clustering algorithm and how it relates to the classification problem of this research.

Pattern representation

Pattern representation is the step in the clustering algorithm that selects the patterns to use and the features of these patterns to use [18]. This step could also decide the number of clusters to use, but for this project this number is unknown and is not decided (the number of threats is unknown).

The patterns to use are already decided by the Intrusion Detection System part of this research; of all the possible network communications, only the actual attacks are selected and presented to the classifier as patterns. This also allows for the idea that the complete research could be modeled as a clustering algorithm; the IDS and Threat profiler are part of the clustering algorithm.

For each pattern (attack), there are many different features that can be selected. Each feature can be of the following types [14]:

1. Quantitative features: e.g.
    a) Continuous values (e.g., weight);
    b) Discrete values (e.g., the number of computers);
    c) interval values (e.g., the duration of an event).
2. Qualitative features;
    a) nominal or unordered (e.g., color);
    b) ordinal (e.g., military rank).

There are however, no theoretical guidelines how to choose the features to use, and most algorithms conveniently assume that the best features are just available for the clustering step [18]. To itemize a list of possible features to extract from the patterns, a detailed look on the patterns is required. Each pattern (attack) consists of one or more IP headers, one or more TCP headers, one HTTP request, one time stamp, one IDS signature and the matching data. This results in the following list of possible features:

Generic pattern information The pattern generated by the TDS contains some information about the attack

Timestamp The IDS will add a timestamp to each attack, denoting the date, and time with microsecond resolution the attack was detected.

Signature ID The signature id denotes which signature matched on the packet and triggered the alert.

Matching data The matching data are the part of the network traffic that matched the signature.

IP [52] The IP packet is used to transmit information from one computer to another.

Version Normal communications over the Internet use IP version 4. In the future, IP version 6 will also be used, but this is outside the scope of this research.

Header length The IP header length field is calculated from the packet to contains the length of the IP header.

Differentiated Services [49] The IP differentiated Services field used to be the Typo of Service field but is replaced in 1998. Now, it is used to classify network traffic so network operators can give certain classes (for example VOIP) priority over others.

Total length The IP total length field is calculated from the packet to contains the total length of the IP packet.

IP identification The IP identification field is a random value for each IP packet.

Reserved flag The IP reserved flag field is always set to zero.

Don't fragment flag The IP don't fragment flag is used to denote that the packet should not be fragmented.

More fragments flag When a packet is fragmented, the IP more fragments flag denotes that more fragments are following.

Fragment offset When a packet is fragmented, the IP fragment offset field denotes the offset of this fragment in the complete packet.

Time to live Routers use the IP time to live field to prevent loops. Each router forwarding the packet will decrease this field by one. When this field reaches zero, the packet is discarded.

Protocol The IP protocol field denotes the protocol encapsulated in the IP packet. In this research, the value of this field is always six (TCP)

Header checksum The IP header checksum field is calculated from the IP header to detect corruption.

Source address The IP source address denotes the sender of the packet. This information is also provided by the IDS for each pattern.

Destination address The IP destination address denotes the recipient of the packet. This information is also provided by the IDS for each pattern.

IP options The IP options field is used to include various options like time stamp and MTU information.

IP data In this research, the IP data will always be a TCP packet.

TCP [53] A TCP packet is used for a TCP connection.

Source port The TCP source port denotes the port used by the sender.

Destination port The TCP destination port denotes the port used by the receiver.

Sequence number The TCP sequence number is a random value for each TCP connection.

Acknowledgement number The TCP acknowledgement number is a random value for each TCP connection.

Data offset The TCP data offset field denotes the offset of the TCP data in the packet.

Explicit Congestion Notification flags [65] The TCP explicit congestion notification flags are used to signal network congestion.

Control bits The TCP control bits (URG, ACK, PSH, RST, SYN, FIN) are used to control the TCP connection (setup, data transmission, and teardown).

Window The TCP window field is used to denote the size of the receiver window. This field is used to inform the recipient how many data can be transmitted before an acknowledgement is required.

Checksum The TCP checksum field is calculated from the packet and is used to detect corruption.

Urgent pointer The TCP urgent pointed denotes data that are considered urgent.

TCP options The TCP options field is used to include various options like window scaling, selective acknowledgement and timestamp information.

TCP data In this research, the TCP data will always be a HTTP request

HTTP request [41, 45] The HTTP request is a request from a web browser to a web service.

Method The HTTP method field denotes the action performed on the service. Most often, this is either GET or POST.

URI The HTTP Uniform Resource Identifier denotes the resource to access on the service.

Version The HTTP version field denotes the HTTP version used. This is normally 1.1 but values of 1.0 are also used.

Headers The HTTP headers contain additional information about the request like the browser version, the HTTP cookies and the referring web page.

Body The HTTP body is only used for HTTP methods that upload data to the service. This could be a file, or the content of a form submitted using the POST method.

Of these possible features, most features are calculated based on the packet or the connection and should not be used to identify the attacker. These features could be used to identify attacks, but this is the responsibility of the IDS component and not of the classifier. This leaves only the following features to possibly use for the classifier:

- Generic pattern information
  - Timestamp
  - Signature ID
  - Matching data
- IP
  - Source address
  - Destination address
  - IP Options
  - IP data
- TCP

  - Source port
  - Destination port
  - TCP options
  - TCP data
- HTTP request
  - Method
  - URI
  - Version
  - Headers
  - Body

For this research, the source IP address is used based on the assumption that threats do not change IP address during their attacks and that IP addresses are not used by multiple threats. This assumption does not hold in every case, but will provide a good starting point for this research. Future research could extend the threat classifier to use multiple features and thereby improve the accuracy of the classifier.

Clustering or Grouping

When the features of patterns are selected, the actual clustering or grouping of patterns is performed. There are many different clustering algorithms, each with their own properties. For this research, the following requirements and limitations are posed on the clustering algorithm:

1. The number of clusters (threats) is unknown;
2. One feature (the source IP address) is present for each pattern (attack).
3. Each pattern (attack) must be grouped into one cluster (the threat). Future research could allow for attacks to belong to multiple threats; each with a certain confidence value indicating a confidence of this attack belonging to this threat.
4. Each cluster (threat) is unrelated to other clusters (threats). Future research could prove this assumption to be false and find clusters within clusters (threats working together). For this research, the threats are either grouped into multiple separate clusters or they are grouped together in one cluster, depending on the source IP address that is used.

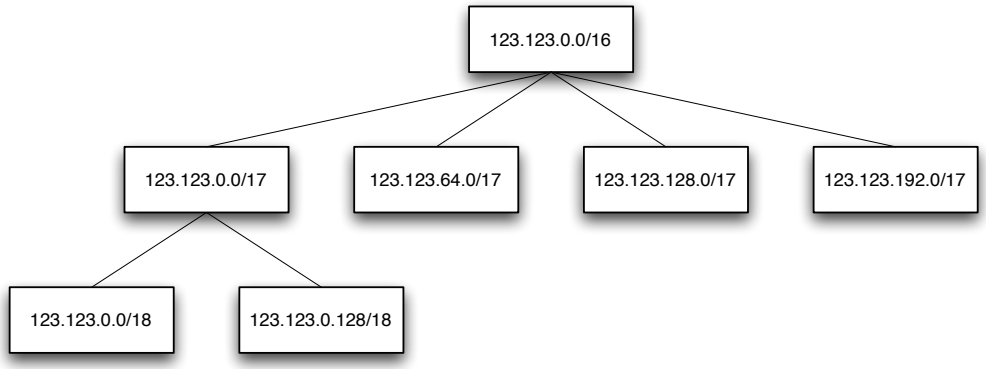

Figure 4.2: Hierarchical clustering algorithm

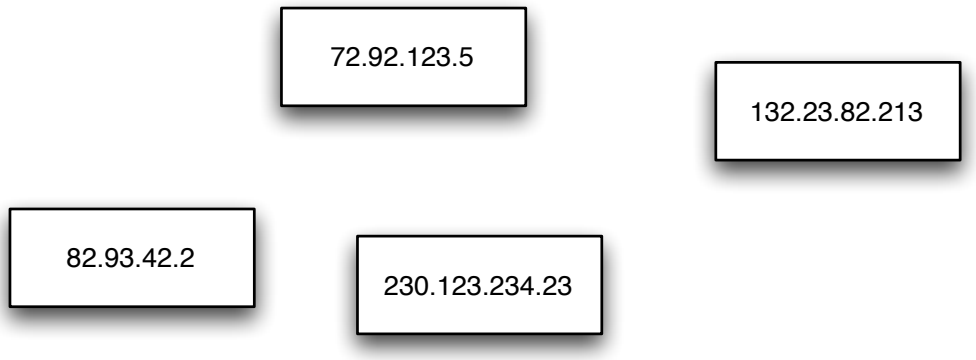

Figure 4.3: Partitional clustering algorithm

Clustering algorithms are split in hierarchical and partitional algorithms. The hierarchical algorithms organize the patterns into a tree (a nested series of partitions [18]) as illustrated in Figure 4.2, where the partitional algorithms create a partition without nesting as illustrated in Figure 4.3.

For this research, each cluster represents the attacks of one threat. Since the assumption is that each threat uses one IP address, the clustering algorithm is very simple: all threats from one IP address belong to the same cluster.

Evaluation of assumption

An important assumption in this research is that threats use one IP address. This section discusses the validity of this assumption.

Literature states that threats can use a different IP address for each step of their attack [2]. However, practical experience in this field tells that this is not the case for real hackers, but only for automated worms. Hackers will use a system they earlier compromised as source of their attack, but are unlikely change systems (and therefore IP address) during the attack.

To evaluate this assumption, a distinction is made between targeted and un-targeted attacks. Targeted attacks involve a threat that is focussing on a specific network. Such a threat is either executing attacks from his own system or operating a set of computers to camouflage the attack. For un-targeted attacks, a worm or virus is circulating the Internet infecting systems at random. Each compromised system then starts attacking other systems. A special case of this is the botnet, where compromised systems become part of a network that is controlled by a person or group of persons.

For the un-targeted attacks by worms, virii and botnets, it could be reasoned that their attacks should be clustered together; this would lead to one threat for each worm, virus, or botnet attacking the network. However, it could also be reasoned that the computers are compromised and should be counted as separate threats; they are each trying to compromise the network. If the TDS would combine the attacks of one worm, virus, or botnet in one threat instead of separate threats for each infected computer, it would change the scenario where, for example, 100 threats perform 5 attacks each, to a scenario where 1 threat performs 500 attacks. This changes the characteristics of the threat to a targeted attack (one threat, many attacks) instead of an un-targeted one (many threats, few attacks). For this thesis, the choice is therefore made to count each compromised computer performing attacks on the network as separate threats.

For targeted attacks, the assumption is that threats will not camouflage their attacks by using multiple systems. When threats do use multiple systems, they are detected as multiple threats performing a few attacks each. It is possible for a threat to use one IP address to perform a few complicated attacks, and several other IP addresses to perform many easy attacks. The system will now detect one threat with an advanced skill level executing a few attacks, and some threats with a beginning skill level executing a few attacks each. In the case of network security monitoring, the security expert is notified of multiple threats attacking the network. When using the system for risk management, the security expert must allow for an error rate in the system; when 10 threats are detected with a beginning skill level and 1 threat with an advanced skill level, it is possible the risk is over-estimated since all threats are actually one. In both scenarios, the threat is detected multiple times instead of just once.

Threats could also share their computer with multiple threats, detected by the system as one threat performing many attacks instead of a few threats

performing a few attacks. In the scenario of network security monitoring, the security experts focus on all activity of the IP address of this threat, and are being more efficient: instead of looking at multiple IP address they only have to look at one. In the scenario of risk management, sharing a computer causes an under-estimation of the number of threats on a network and this error rate should be taken into account when using the system.

In conclusion, targeted threats gain from sharing their computer when the system is used for risk management: the risk is under-estimated. This under-estimation error rate is calculated in chapter 5.

4.4 Threat profiler

The threat profiler component of the Threat Detection System receives a list of threats and their attacks from the threat classifier component. For each threat, a set of concrete properties is assigned. Although it would be pleasant to assign a "motivation" or "intent" property to a threat, this is abstract and cannot be objectively measured. Therefore, only properties that can be objectively measured should be implemented. This research implements the number of attacks detected by system for this threat as on objectively measured property.

The research also implements a subjective property: the skill of the threat. Although this skill property subjective, it is based on objective measurable facts: the attack methods detected for the threat. The interpretation of these facts is subjective and is therefore configurable: a security expert with a different opinion on skill could configure the system to reflect her ideas.

Finally, an abstract property is also implemented: the android property stating whether the threat is a human or a computer program. Abstract properties should not be implemented at all, because they constitute of concepts that cannot be measured. For this research, the android property is implemented because there is a simple measurable fact (how fast are attacks performed) that is a strong indicator whether the threat is a computer (the attacks are performed inhumanly fast) or a human (not all attacks are performed inhumanly fast). By implementing this abstract property, the added value of such properties can be determined.

Intensity class profiler

The intensity profiler defines the attack intensity of a threat, which is an objective fact that can be measured by the system. The NIST standard [34]

uses motivation instead of attack intensity, but because motivation is an abstract interpretation for which no measurement that can be performed by a computer is known, this research cannot provide a motivation profiler. The intensity profiler can measure the number of attacks and the rate of these, and leave the interpretation of this data to the security expert.

The intensity profiler could measure several statistics for each threat that could be used to determine the intensity class of this threat. For example, the total number of attacks is an important metric to identify the motivation of threats. However, since not all threats are using tools to perform a large number of attacks, another important metric is the number of minutes the threat is active attacking the network.

This research could assign the number of attacks and the number of minutes active to each threat. However, to allow security experts to select generic classes instead of a specific number of attacks or minutes, four intensity classes are introduced as illustrated in Thresholds for the intensity classes (one of the thresholds must be met) (Table 4.2). Since these classes are on interpretation of the data, the thresholds are configurable to allow different interpretations by other security experts. The thresholds in this implementation are chosen on personal experience with attack & penetration testing and are explained in more depth in section 4.4.

Table 4.2: Thresholds for the intensity classes (one of the thresholds must be met)

Intensity Class	Number of attacks	Number of minutes active
Intensity 1	> 1	> 1
Intensity 2	> 10	> 5
Intensity 3	> 100	> 25
Intensity 4	> 1000	> 125

Skill class profiler

The skill profiler defines the skill class of a threat, which is a subjective property. The implementation of this profiler is based on a subjective opinion of skill and is therefore configurable by other security experts to match their opinions.

The Intrusion Detection System component can detect generic attack methods and annotates each detected attack with the signature that was triggered which is linked to an attack method. For each attack method, a choice is made whether this method can be used by beginning threats or

only by more advanced threats. All methods listed in the Certified Secure Basic Web Application Scan Checklist [64] are well known and easy to perform, the methods listed in the Certified Secure Advanced Web Application Scan Checklist [62] are either complicated to perform or known by only a few.

The skill profiler is based on the idea that a threat performing an attack method listed in the basic checklist is of at least basic skill; the threat is therefore able to perform any of the attack methods listed in the basic checklist since all methods listed in the basic checklist are well known and easy to perform. If the threat performs an attack listed in the advanced checklist, the threat is either capable of performing a complicated attack method or has knowledge of an attack method that is not widely known. The threat is therefore considered to be able to perform any of the attack methods listed in the advanced (or basic) checklist. The concepts of "basic", "advanced", "easy", "complicated", "known", "not widely known" are all subjective and implemented using the interpretation of the Certified Secure checklists. If a security expert disagrees with these interpretations, she can easily change to the algorithm to reflect a different set of classes.

The relationship between the signatures, the checklist items, and the resulting skill class is presented in Table 4.3. This information is stored in a relational database that can be queried by the skill class profiler. The security expert using the TDS must consider the hierarchy of the classes; a threat able to perform an attack from the "advanced" class, is also able to perform an attack from the "basic" class.

The skill property of a threat is therefore calculated by locating the checklist of each signature. When a threat performs only attacks listed in the basic checklist, the skill of the threat is defined as "basic". A threat performing at least one attack in the advanced checklist is defined as "advanced". It is not possible for a threat to have a "none" skill class, because any threat detected by the system has executed at least one attack and is therefore defined with at least a "basic" skill level.

Android profiler

The android profiler defines whether the threat is a computer or a human. This is an abstract property, and such properties cannot be accurately implemented in a computer program. The android profiler is implemented to test the accuracy if such a property implemented.

For this property, the profiler is implemented based on the observation of inhuman behavior. Threats that manually execute attacks are likely to think before the next request and have to manually type the attack.

Table 4.3: Relationship between signatures, checklist items, and skill classes

Signature	Checklist item	Skill class
1120001	Check for default and predictable accounts	basic
1120002	Check for default and predictable accounts	basic
1120003	Check for default and predictable accounts	basic
1120004	Check for default and predictable accounts	basic
1120005	Check for default and predictable accounts	basic
1120006	Check for default and predictable accounts	basic
1210001	Check for filename injection / path traversal	basic
1210002	Check for filename injection / path traversal	basic
1220001	Check for SQL injection	basic
1220002	Check for SQL injection	basic
1220003	Check for SQL injection	basic
1230001	Check for cross site scripting	basic
1230002	Check for cross site scripting	basic
1230003	Check for cross site scripting	basic
1230004	Check for cross site scripting	basic
1230005	Check for cross site scripting	basic
1230006	Check for cross site scripting	basic
1230007	Check for cross site scripting	basic
1230008	Check for cross site scripting	basic
1240001	Check for system command injection	basic
1240002	Check for system command injection	basic
1310001	Check for uploading of (dynamic) scripts	basic
20210001	Check for extraneous files in document root	advanced
20340001	Check for accessible CVS/SVN directories	advanced
20340002	Check for accessible CVS/SVN directories	advanced
20370001	Check for accessible non-parsed dynamic scripts	advanced
20650001	Check for brute-force username enumeration	advanced
20650002	Check for brute-force username enumeration	advanced
20710001	Check for double decoding of headers / parameters	advanced
20710003	Check for double decoding of headers / parameters	advanced
20710004	Check for double decoding of headers / parameters	advanced
20710005	Check for double decoding of headers / parameters	advanced
20760001	Check for XSL(T) injection	advanced
20770001	Check for SSI injection	advanced
20790001	Check for dynamic scripting injection	advanced
20810001	Check for XML external entity parsing	advanced
20820001	Check for XML external DTD parsing	advanced
2091001	Check for uploading outside of intended directory	advanced
2091002	Check for uploading outside of intended directory	advanced
2094001	Check for uploading of configuration files	advanced
2094002	Check for uploading of configuration files	advanced

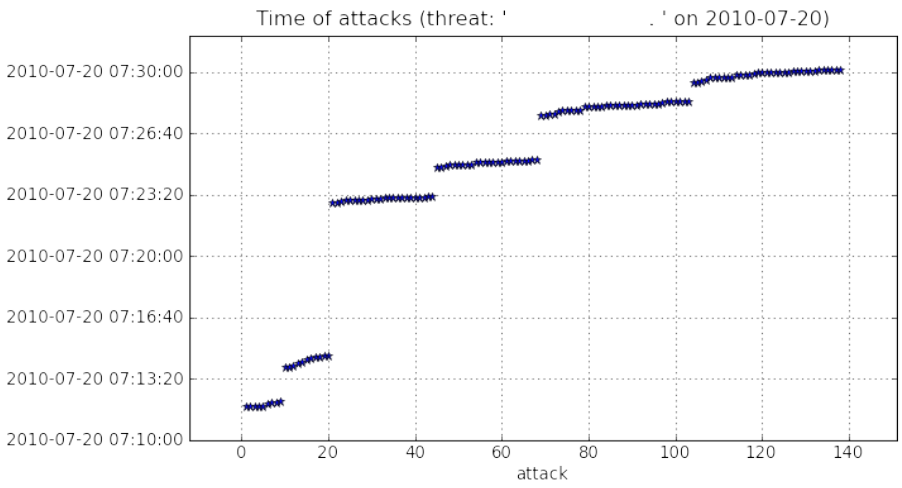

Figure 4.4: Timing of attacks performed by a computer. For this illustration, data of the Real network experiment, as described in the Benchmark and results chapter, is used.

Therefore, when the interval between two attacks is too small, it must be have been a computer program performing these attacks. The algorithm to define the android therefore uses a set of threshold values. Different thresholds for attacks within one second, within one minute, and within one hour are set.

A threat is identified being a computer program when all attacks are performed at inhuman speed, as defined by the thresholds. The timing of the attacks performed by a computer program is illustrated in Figure 4.4. When a threat performs at least some manual attacks, the threat is identified as being a human being. Such a threat could perform only manual attacks (illustrated in Figure 4.5) or a combination of manual attacks and automated attacks (illustrated in Figure 4.6).

In this profiler, a threat is considered human when the number of attacks that are not part of computer attacks is above a certain threshold. For example, if the threat executes only a tool, then all attacks are within the computer thresholds and no human attacks will be counted. If the threat then tries some attacks himself, they will count towards the human threshold.

The exact thresholds are configurable, but they represent properties of the actual threats and therefore will not change much between different networks. This means that the thresholds can be set once using an experiment. The values can then be used by many different networks, under the assumption that the threats will not change.

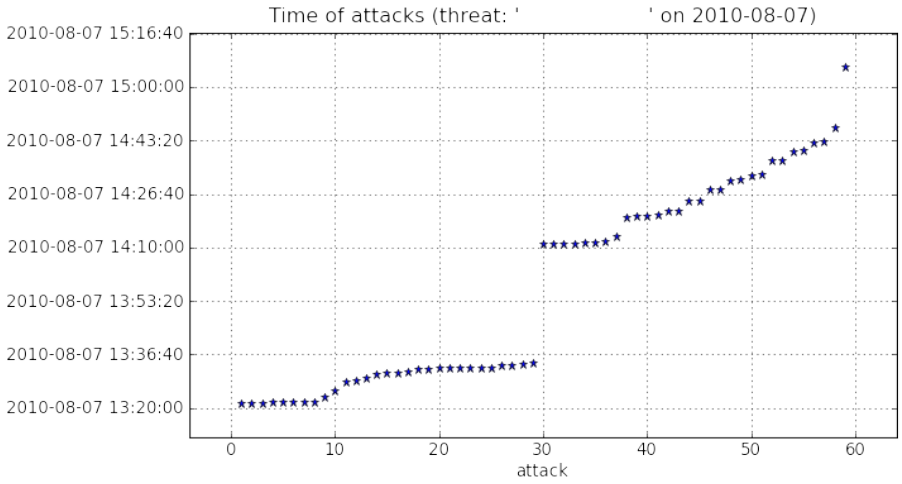

Figure 4.5: Timing of attacks performed by a human. For this illustration, data of the Hacker experiment, as described in the Benchmark and results chapter, is used.

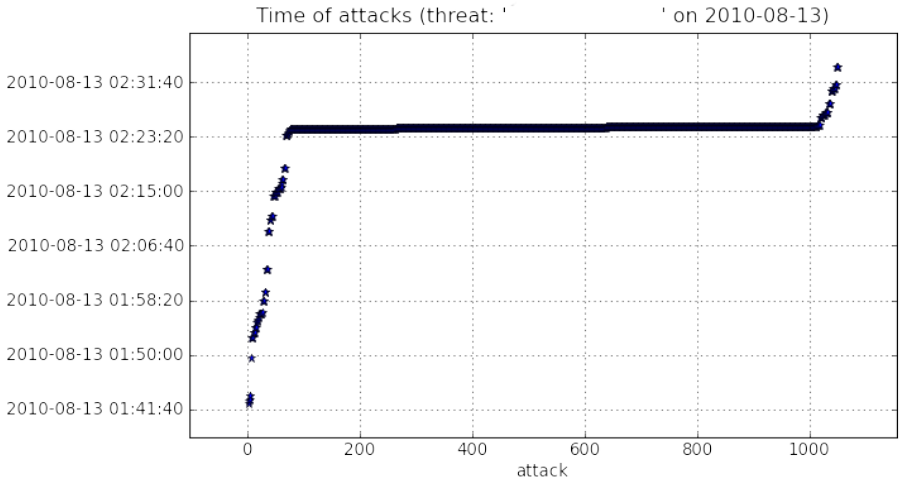

Figure 4.6: Timing of attacks performed by a human who also uses some computer tools. For this illustration, data of the Hacker experiment, as described in the Benchmark and results chapter, is used.

Usage

The following definition for the classes can be used when using the skill, android, and intensity property in risk management and security monitoring:

basic skill class The threat is capable of executing the attacks against vulnerabilities that are listed in the Certified Secure Basic Web Application Scan Checklist [64].

advanced skill class The threat is capable of executing the attacks against vulnerabilities that are listed in both the Certified Secure Basic Web Application Scan Checklist and the Certified Secure Advanced Web Application Scan Checklist [62].

Computer class The threat is using a computer programs to perform attacks. The security expert could select the computer class when exploiting the vulnerability can be performed by a tool.

Human class The threat is an actual human beings using a browser or telnet to manually attack systems. When a threat is (also) performing manual attacks, the chances of exploiting a vulnerability are generally much higher as they can think about what happens and alter their approach; automated tools will fail when the system is not responding as expected. The security expert must select the human class when assessing a vulnerability that requires actual thinking and manual steps (for example requiring a captcha [39]).

intensity 1 class The threat has executed at least one attack on the network.

intensity 2 class The threat has executed at least 10 attacks on the network or is active for 5 minutes. This means that the threat did not try just one thing and left when it didn't work, as is usual for worms and virii. The threat will not be able to find all vulnerabilities, but there is a chance that a vulnerability is found and exploited.

intensity 3 class The threat has executed at least 100 attacks on the network or is active for 25 minutes. This threat could be interpreted as being serious about compromising the network and is likely to be able to exploit vulnerabilities that are simple for his skill class.

intensity 4 class The threat has executed at least 1000 attacks on the network or is active for 125 minutes. This threat could be interpreted as being very serious about compromising the network and is likely to be able to exploit vulnerabilities that are difficult for his skill class.

When using the TDS to improve the estimation of a vulnerability as part of risk management, the security expert should locate the attack method required to exploit the vulnerability in either the basic or the advanced checklist. This checklist then corresponds to the skill class that is required for threats to be capable of exploiting this vulnerability. Because of the hierarchy in the classes, when the vulnerability is listed in the basic checklist, the threats of the advanced skill must also be included in the query. If the vulnerability is not listed on either of the checklists (as they cannot be complete), the security expert has to decide on which checklist the vulnerability belongs.

Then, the security expert should determine if the vulnerability can only be exploited by a human. An example of this would be the case where the threat must register for an account and then log in to the system; these steps are too complicated for an automated tool and therefore, the human class is selected. In all other cases, the security expert selects the computer class and human class for this vulnerability, as both computer programs and humans can exploit the vulnerability.

Finally, the security expert selects the intensity class that is required for this vulnerability. The intensity class should correspond with the number of attacks, and the amount of time required to identify the vulnerability. Some vulnerabilities can be identified in one attempt, for example cross site scripting. Other vulnerabilities require a few attempts, for example, SQL injection. How many attacks are required to exploit the vulnerability is irrelevant, since the threats will first identify if a vulnerability exists before attempting to exploit it. For the metrics, it is important to also count the threats looking for this vulnerability, even when they don't find it. As with the skill and android class, the classes of higher intensity should also be selected: instead of selecting just intensity 2, also intensity 3 and 4 must be selected.

When the appropriate classes are selected for the vulnerability, the security expert can consult the TDS. The query to the TDS could for example be 'how many threats of (basic skill or advanced skill) and human and (intensity 2 or intensity 3 or intensity 4) were identified on the network in the last 7 days'. This number will not be the exact chance that a threat will exploit the vulnerability but is a number that can be used in the estimation [31].

If a security expert uses the TDS for network security monitoring, the skill, android, and intensity properties provide information about the active threats. By referencing the checklists, the possible attacks of the threats become clear and a decision to further investigate this threat can be made. Because the attacks of uninteresting threats are grouped together, the security expert will spend little time investigating those uninteresting events.

Chapter 5

Benchmark and results

In this chapter, the testing methodology and results are presented for a final evaluation of the system. The results show if the research meets the problem definition:

> *Design a system to detect and profile threats on a computer network to improve risk management and network security monitoring.*

An important measure to evaluate IDSs, is the rate of false positives and false negatives. Ideally, these measurements are performed using a labelled data set that is representative for the purpose of the system. In this case, a data set that contains attacks of threats which are annotated (labelled), could determine the number of users that are falsely detected as being a threat (false positive rate) and the number of threats that are not detected at all (false negative rate). The only publicly available data set that is current and provides labels for each threat and attack, contains only the netflow information and not the full contents of the network packets [32] and is therefore not suitable for this research.

To evaluate if the system provides meaningful information to security experts performing risk management and network security measurement, the results of the system are analyzed in two scenarios. First, a test is executed by placing the system in a real network for 4 days. For this network the threats and their attacks are unknown, but all detected threats can be manually evaluated and false positives can be found, providing an indication of the false positive rate of the system. In the second test, the system is placed in a network containing only threats. The network hosts a web hacking challenge and is therefore not representative for real world scenario's (the attacks are similar, but in the real world the threats have

no permission to attack the sites and therefore could behave differently). Since the threats are known and fill out a questionnaire, this test allows to test the false negative rate of the system which is not possible in a scenario that is truly representative for the real world containing unknown threats and unknown attacks.

5.1 Testing methodology

Both tests consist of a number of steps to ensure the quality and accuracy of the results. In this section, the steps are shortly described.

1. Basic information about the network (number of hosts, number of websites, and bandwidth information) is documented.
2. A choice is made for online or offline analyses of the network traffic.
    a) For online analysis (test 1), the TDS is installed on a system that has access to the network traffic (for example using a SPAN port).
    b) For offline analysis (test 2), a full network trace for tcp port 80 is created using tcpdump[1].
3. For the period of collecting information, the system is regularly verified to be functioning properly.
4. After a documented period of time, the network monitoring is closed. For offline analysis, the trace is now transferred to the TDS running on a standalone computer which will process it.
5. The number of attacks detected by the IDS component of the system is documented, along with a breakdown in number of attacks per signature.
6. For test one, each attack is manually verified to be a true attack. The captured trace can provide more information and context to each attack. The false positive rate of the IDS component of the system is now calculated and documented along with the actual false positives and when possible a reasoning for this error.
7. For both tests, the performance of the threat classifier is discussed. This cannot be 100% accurate in the first test, since no information is known about the source IP addresses of the threats. In the second

[1] tcpdump -i interface -s 0 -w trace port 80

test, the IP addresses is recorded for each threat and this information can be used to evaluate the classifier.

8. For both tests, the properties that are assigned to threats are discussed. As in the previous step, this cannot be 100% accurate since no information about the exact properties of each threat is known in the first test.

9. For test one, the number of threats for which only false positive attacks are found, is documented as the false positive rate of the complete system. The reason why these threats are found as such is already documented in the step documenting the false positive rate of the IDS component.

10. For test two, the number of threats that have been recorded to complete a hackme challenge, but are undetected by the system, are documented as the false negative rate of the complete system. If possible, the reason why this threat wasn't detected is documented as well.

11. For test two, the correlation between the skill of the threats determined by the questionnaire and the skill of the threat determined by the system is calculated.

5.2 Real network experiment

The first test is the baseline scenario. During the development of the system, it was placed within a real network hosting around 100 websites for the duration of 28 days to get a general idea of the working of the system and setting the system parameters. In this experiment, the system is placed within the same network for four days to evaluate the performance of the TDS in a real network.

IDS results

The IDS component of the TDS is responsible for detecting the actual attacks on the network. The false positive and false negative rate of this component has direct influence on the performance of the TDS. However, the false negative rate of this component should not have that much impact on the performance of the TDS, as long as some attacks are detected for each threat.

In this scenario, a total of 1343 attacks were detected by the IDS of which 540 were false positives. This is a false positive rate of 0.40. In Real network

experiment: Signatures, True Positives and False Positives (Table 5.1), a breakdown of is given of true and false positives for each IDS signature.

Table 5.1: Real network experiment: Signatures, True Positives and False Positives

signature	true positives	false positives	Checklist item
1210001	221	540	Check for filename injection / path traversal
1220001	10	0	Check for SQL injection
1220002	168	0	Check for SQL injection
1220003	128	0	Check for SQL injection
1230001	71	0	Check for cross site scripting
20210001	184	0	Check for extraneous files in document root
20650001	8	0	Check for brute-force user name enumeration
20790001	13	0	Check for dynamic scripting injection

The false positives generated by the IDS are all originating from one IP address, using "../" in the Referrer header field as illustrated in Listing 5.1. As previously documented section 4.2, these requests aren't meant as filename injection attacks, but could have been attacks in another context. No other false positives are detected by the IDS.

```
GET /anonymized HTTP/1.1
Accept: text/plain,text/html,*/*;q=0.3
TE: trailers
Host: anonymized
Referer: ../../
User-Agent: Mozilla/4.0 (eknip/1.60)
Connection: TE
```

Listing 5.1: One threat performed 540 requests that were falsely detected as an attack. The threat was indexing large parts of a website and used ../ in the Referer header field. This field is used to communicate to the web server which page was visited previously to this page and allows websites to know which other websites link to them. Using ../ in this field is highly unlikely, but the threat in this case did not try to attack the network.

Threat classifier and profiler results

The 1343 attacks that are detected by the IDS are clustered by the threat classifier in 47 threats. A breakdown of threats and their properties is provided in Table 5.2. For each threat, the accuracy of the threat classifier and threat profiler is evaluated:

Table 5.2: 47 threats and their properties

id	skill	android	intensity	incorrect classification
1	basic	computer	intensity 3	
2	advanced	computer	intensity 3	
3	basic	computer	intensity 3	
4	basic	computer	intensity 1	
5	basic	computer	intensity 1	
6	basic	computer	intensity 1	
7	basic	computer	intensity 2	
8	basic	computer	intensity 2	
9	basic	computer	intensity 2	
10	basic	computer	intensity 1	
11	basic	computer	intensity 1	
12	basic	computer	intensity 1	
13	basic	computer	intensity 1	
14	basic	computer	intensity 1	
15	basic	computer	intensity 1	
16	basic	computer	intensity 1	
17	basic	computer	intensity 1	
18	basic	computer	intensity 1	
19	basic	computer	intensity 1	
20	basic	computer	intensity 1	
21	basic	computer	intensity 1	
22	basic	computer	intensity 1	
23	basic	computer	intensity 1	
24	basic	computer	intensity 1	
25	basic	computer	intensity 1	
26	basic	computer	intensity 1	
27	basic	computer	intensity 1	
28	basic	computer	intensity 1	
29	basic	computer	intensity 2	
30	basic	computer	intensity 2	
31	basic	computer	intensity 2	
32	basic	computer	intensity 2	
33	basic	computer	intensity 2	
34	basic	computer	intensity 2	
35	advanced	computer	intensity 1	
36	advanced	computer	intensity 1	
37	advanced	computer	intensity 1	
38	basic	computer	intensity 1	
39	basic	computer	intensity 1	
40	basic	computer	intensity 1	
41	basic	computer	intensity 1	
42	basic	computer	intensity 2	
43	basic	human	intensity 1	wrong android

Table 5.2: 47 threats and their properties

id	skill	android	intensity	incorrect classification
44	basic	computer	intensity 1	
45	basic	computer	intensity 1	
46	basic	human	intensity 1	wrong android
47	basic	computer	intensity 1	

- The 540 false positives are represented by 1 threat (id 1) that isn't a threat and therefore a false positive, as discussed in section 5.2 and illustrated in Listing 5.1. The profile of this threat consists of the basic skill class and an intensity of 3 (a threat being serious about compromising the system). The threat is listed as computer, which is correct since it automatically downloads many pages using some tool. Security experts monitoring this network should investigate this threat and then conclude this threat is a false positive.

- One threat (id 2) that is listed with advanced skill and intensity 3, performed 292 attacks by using a tool on multiple occasions (10 times in 2 days). The intensity class is appropriate for this behavior, and the tool is able to perform advanced attacks like searching for extraneous files on the website. This threat is listed as a computer, which is correct since only automated attacks are performed. The threat profile for this threat is therefore accurate.

- Four threats (ids 3-6) performed many SQL injection attacks illustrated in Listing 5.2, one of which is listed as intensity 3 for performing 104 attacks and listed with a basic skill for only performing attacks listed in the basic checklist. All threats are listed as computers, which is correct. This intensity and skill class are appropriate for this threat.

- There are 22 threats (ids 7-28), all part of a bot-net that perform an identical filename injection attack as illustrated in Listing 5.3. Three of these threats are profiled with intensity class 2, because they performed more than 10 attacks. All threats are profiled to be of basic skill, which is correct since they perform only filename injection attacks. Also, all threats are listed being a computer which is also correct. These attacks are the beginning of a string of attacks named "Local File Inclusion", and detecting such attacks in their first stage could be useful for network security monitoring.

- Interestingly, six threats (ids 29-34) performed similar SQL injection attacks (illustrated in Listing 5.4) simultaneously. The number of attacks for each host is 30, which could a method of camouflaging the attacks of one threat. Instead of one threat with intensity class 3, this threat is reported as 6 threats, each with intensity class 2. Apparently, camouflaging

is happening on real networks. All threats are listed as being computers, which is correct.

- Three threats (ids 35-37) performed an advanced dynamic scripting injection attack (illustrated in Listing 5.5) and therefore have the advanced skill property. These threats executed a few attacks (6, 6 and 1 attacks per threat) and therefore are listed as intensity 1. Also, these threats are correctly listed as computers since this attack is performed by a tool.

- Two threats (ids 38 and 39) performed a cross site scripting attack, as illustrated in Listing 5.6. These threats are systems of Yahoo and Scoutjet, which suggests that some website embedded an attack in their links. This could be a novel way of performing attacks anonymously. The intensity (1), skill (basic) and android (computer) classes for these threats are appropriate.

- An interesting event is illustrated in Listing 5.7 caused by two threats (ids 40 and 41). In this case, a threat performed a normal SQL injection attack and Google repeated this attack within seconds. The reason for this behavior is the use of Google analytics on the target website. Apparently, when Google analytics is included on a URL that is not yet known by Google, it will be downloaded by Google. This could also be an interesting new way of performing attacks anonymously. The intensity (1) and skill (basic) classes for these threats are appropriate. The android (computer) class is also acceptable, since only one attack is not enough to determine if the threat is a human.

- Three threats (ids 42-44) performed a filename injection similar to threats 7-28, as illustrated in Listing 5.8. These threats however, are part of a different bot-net since their requests are slightly different. For one of these threats, the profiler incorrectly assigns the human class. The skill (basic) and intensity (2, 1 and 1) classes are correct.

- One threat (id 45) performed sql injection attacks on a site using an automated tool (illustrated in Listing 5.9). The skill (basic), android (computer) and intensity (1) classes are correct for this threat.

- One threat (id 46) performed a filename injection attack as illustrated in Listing 5.10, similar to threats 7-28 and threats 42-44. This threat is also different in the exact attack, and is incorrectly assigned the human class. The skill (basic) and intensity (1) classes are correct.

- Finally, one threat (id 47) performed a filename injection attack as illustrated in Listing G.1 which is, again, similar to threats 7-28, 42-44 and 46. This threat is correctly assigned the basic, computer and intensity (1) classes.

This experiment shows that the system is capable of detecting threats on a real network. The false positive rate of the system during this experiment is 1 out of

47 (2.1%). The profiler correctly identified the skill and intensity class to each threat, and identified the correct android class for all except two threats (4.3%).

```
GET /cmsms/index.php?page=anonymized\%27\%20\%61\%6E
    \%64\%20\%27\%36\%27\%3D\%27\%35 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Accept: */*
Host: anonymized
Cache-Control: no-cache
Cookie: CMSSESSIDfcd6a619=ad57bbc5c6c31bb6b2c3ef38c1fbc09c
```

Listing 5.2: 4 threats performed many SQL injection attacks on different sites using the same sql injection encoding (%27%20%61)

```
GET /?option=com_preventive&controller
    =../../../../../../../../../../../../../../../../proc/self/
    environ\%00 HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: anonymized
User-Agent: libwww-perl/5.832
```

Listing 5.3: 22 threats performed this path traversal attack. By including the /proc/self/environ file, the threat can easily test if the website is vulnerable to this attack

```
ey=1&hit=1&option=com_zoom&itemid=66&page=view&catid=6&pageno
    =1\%20and\%201=2\%20union\%20select\%20CONCAT(0x27,0x7c,0
    x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f
    ,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0
    x7c),CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c
    ),CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c),
    CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c),
    CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c),
    CONCAT(0x27,0x7c,0x5f,0x7c),CONCAT(0x27,0x7c,0x5f,0x7c)
    \%20/* HTTP/1.1
User-Agent: czxt2s
Host: anonymized
Cookie: dd9d28a1e5ca5f12f0e5797c7a4bae3b=
    f6ffff1370072c1402d83d9e74d0e0ef; mosvisitor=1; PHPSESSID=9
    pvv0cqhp9lekph1dul280rvo1
```

Listing 5.4: 6 threats performed this SQL Injection attack (only the matching part of the request is displayed)

```
GET //lib/adodb_lite/adodb-perf-module.inc.php?last_module=
    zZz_ADOConnection\%7B\%7Deval($_GET[w]);class\%20
    zZz_ADOConnection\%7B\%7D//&w=include($_GET[a]);&a=http://
    www.dong69.co.kr/xe/ganyot/test?? HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: anonymized
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 6.0)
```

Listing 5.5: 3 threats performed this dynamic scripting injection attack. This attack includes a script hosted on www.dong69.co.kr that contains the actual commands being executed

```
GET /anonymized/22'></script></p> HTTP/1.1
Host: anonymized
User-Agent: Mozilla/5.0 (compatible; ScoutJet; +http://www.
    scoutjet.com/)
Accept-Encoding: gzip
Accept: */*
Connection: close
```

Listing 5.6: Two threats (yahoo and scoutjet) perform this Cross Site Scripting attack, probably because some website included a link with this attacks. This is an interesting attack since threats are able to use search engines to perform their attacks

```
GET /?ID=1\%20OR\%201=2 HTTP/1.1
Host: anonymized
Connection: Keep-alive
Accept: */*
User-Agent: Mediapartners-Google
Accept-Encoding: gzip,deflate
```

Listing 5.7: Two threats (including google) performed this SQL Injection attack

```
GET /components/com_joomlalib/standalone/stubjambo.php?baseDir
    =../../../../../../../../../../../../../../proc/self/
    environ\%00 HTTP/1.0
Host: anonymized
Accept: */*
User-Agent: Mozilla/5.0
```

Listing 5.8: 3 threats performed this filename injection attack which is similar to the previous filename injection attack but uses different headers

```
GET /index.php?id=84\%20aND\%208\%3D8 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Accept: */*
Host: anonymized
Cache-Control: no-cache
Cookie: PHPSESSID=ogg6qb008ja019as7jbvi0dc53; fe_typo_user=10
    ad246fa5d01227bc1130f8fe90fd42; anonymized
```

Listing 5.9: 3 threats performed this filename injection attack which is similar to the previous filename injection attack but uses different headers

```
GET /index.php?page=../../../../../../../../proc/self/environ
    \%00 HTTP/1.0
Host: anonymized
Via: 1.1 cacheadllink-1:33127 (Lusca)
X-Forwarded-For: 187.17.191.168
Cache-Control: max-age=2419200
```

Listing 5.10: One threat performed this filename injection attack which is similar to the previous filename injection attack but uses different headers

```
GET /index.php?module=PostWrap&page=./../../../../../../../../
    etc/passwd HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: anonymized
```

Listing 5.11: One threat performed this filename injection attack which is similar to the previous filename injection attack but uses different headers

5.3 Hacker experiment

To evaluate the ability of the TDS to identify the skill of threats, an experiment is conducted by inviting hackers to compromise a website and then complete a questionnaire to identify their skill level. For each hacker, the skill level identified by the questionnaire is compared to the skill level as reported by the TDS, to calculate the correlation between the two.

Question

The main question of this experiment is whether the TDS can accurately profile the threats:

> *What is the correlation between the skill level detected by the TDS and the skill level determined by a questionnaire?*

Hypothesis

The hypothesis of this test is:

> *If hackers are invited to attack a computer that is monitored by the TDS and complete a questionnaire to assess their skill, then there is a strong, positive correlation between the two, which is statistically significant.*

Experiment

This experiment consists of the following steps:

1. The link to this experiment is posted on hacker forums and asks professional penetration testers to participate.
2. Participants are provided with a description of the experiment.
3. Participants are provided with the following guidelines:
    - You are invited to hack a website (named "Best Gadgets") we have set up for this experiment. You are only allowed to hack "Best Gadgets" and should use only web-attacks (so no denial of service attacks please).
    - The website includes both easy and difficult vulnerabilities you can try to exploit. It is ok if you find only a few of these vulnerabilities or none at all.
    - While you are hacking the sites, it would be helpful if you keep some notes on the things you have tried and the things you have found. Later in the questionnaire, we will ask you what types of attacks you have tried and if you succeeded in them.
    - When you want to stop hacking (anywhere from a few minutes to a few hours is ok), just press the "Finish hacking" button. You are then asked some questions (in a small questionnaire) about your hacking experience and the things you have tried and found on the website.
    - The information you provide in the questionnaire and the attacks you perform on the hacker challenge will only be used in scientific research. Only anonymized and aggregated data and anonymized examples of interesting attacks will be published.
    - We want to correlate your hacking attacks with the answers you provide on the questionnaire. Therefore we will send you a link to "Best Gadgets" that includes an identifier.
    - If you like to be informed on the results of this experiment, you can state this in the questionnaire.
4. Participants provide their e-mail address and receive a link to the website containing a unique identifier.
5. Participants continue to the website containing the following vulnerabilities
    a) Accessible non-parsed dynamic script in login.php.bak
    b) Cross Site Scripting in the 'leave a message' box
    c) Filename Injection Attack in the page parameter of the URL

d) SQL Injection in the error parameter of the URL

e) XML External Entity Attack in the XMLRPC-Ajax communication

f) XSLT Injection Attack in the search field

6. Participants have a button at the bottom of the screen "finish hacking" to continue with the questionnaire of the experiment.

7. Participants answer a questionnaire containing the following questions

   a) Demographic questions

      i. What is your gender? (male/female)

      ii. In what year were you born?

      iii. What country are you from?

      iv. What is your highest academic degree? (High School Diploma, Bachelor's Degree, Master's Degree, Doctorate, Other (please specify))

   b) Skill quesions

      i. How would you rate your hacking skill? (1-5 rating: 1=very basic skill, 5=very advanced skill)

      ii. Please describe your training/certificates in computer security (if any)

      iii. How would you rate your ability to perform attacks on the following security problems? (1-5 rating: 1=I never heard about this attack, 2=I have heard about the attack, but cannot perform it, 3=I can perform a basic version of this attack, 4=I can perform different forms of this attack, 5=I know all there is of this attack.)

      iv. Accessible non-parsed dynamic scripts: an attack where the hacker tries to find backups of scripts that are left by editors

      v. Cross Site Scripting: an attack where the attacker can inject her HTML code (including Javascript) in the website

      vi. Filename Injection or Path Traversal: an attack where the attacker fool the system to open different files, also in different directories

      vii. SQL Injection: an attack where the attacker can execute her own database statements, or alter the normal statements

      viii. XML External Entity: an attack where the attacker can open files on the system by using the XML External Entity feature

      ix. XSL(T) Injection: an attack where the attacker can inject her own XSL statements, or alter the normal statements.

   c) Did you try the following attacks on the website ("Best Gadgets")? (1=I did not try this attack, 2=I tried the attack briefly, but it didn't work, 3=I tried the attack for a while, but it didn't work, 4=I tried the attack and it worked, 5=I used this attack to compromise the system.)

i. Accessible non-parsed dynamic scripts: an attack where the hacker tries to find backups of scripts that are left by editors

ii. Cross Site Scripting: an attack where the attacker can inject her HTML code (including Javascript) in the website

iii. Filename Injection or Path Traversal: an attack where the attacker fool the system to open different files, also in different directories

iv. SQL Injection: an attack where the attacker can execute her own database statements, or alter the normal statements

v. XML External Entity: an attack where the attacker can open files on the system by using the XML External Entity feature

vi. XSL(T) Injection: an attack where the attacker can inject her own XSL statements, or alter the normal statements.

d) Final thoughts

i. Would you like to be informed on the results of this experiment? (Yes/No)

ii. Do you have any comments or thought you would like to share?

8. The skill level for each participant is identified using the questionnaire based on the skill level definition provided in section 4.4. If the participant said she performed an advanced attack (accessible non-parsed dynamic scripts, XML External Entity, XSL(T) Injection), her level is defined as "advanced". If the participant said to have performed only one or more basic attacks (Cross Site Scripting, Filename Injection or SQL Injection), her level is defined as "basic". When no attacks are performed, her level is defined as "none".

9. The skill level for each participant is determined using the TDS.

10. The correlation between the two skill levels for the participants is calculated using Spearman's Rank Order Correlation Coefficient.

Results

The experiment is executed from 2010-08-07 14:40:43 to 2010-09-01 12:05:12. During this time, the questionnaire was submitted 95 times. Using control questions in the questionnaire, 30 participants are filtered because they provided conflicting information or did not complete the questionnaire. An example of conflicting information is a participant stating that she had no knowledge of performing SQL injection attacks but also stating that she performed an SQL injection attack. Another three participants are filtered because they submitted the questionnaire multiple times. This leaves 59 participants that provided clean data.

Table 5.3: Number of male and female participants

Gender	#	fraction
male	58	0.98
female	1	0.02

The 59 participants are a random selection, since the experiment was announced on different public mailing-lists and forums. Table 5.3 to Table 5.6 illustrate the demographic properties of the data set.

Table 5.4: Year of birth of participants

Year of birth	#	fraction
1970	1	0.02
1973	1	0.02
1974	1	0.02
1976	2	0.03
1977	2	0.03
1978	2	0.03
1979	1	0.02
1980	2	0.03
1981	1	0.02
1982	1	0.02
1983	2	0.03
1984	4	0.07
1985	3	0.05
1986	7	0.12
1987	5	0.08
1988	2	0.03
1989	4	0.07
1990	5	0.08
1991	5	0.08
1992	2	0.03
1993	3	0.05
1995	1	0.02
1996	2	0.03

Using a unique identification number for each participant, which they received in their e-mail, it is possible to track the IP addresses of each participant. Using this data, it can be concluded that no IP addresses are used by two or more participants. Furthermore, there are 12 participants that used 2 or more IP addresses, but only one of those participants performed attacks from 2 different IP addresses. Within this data set, this participant represents 1.7% of the data

Table 5.5: Education of participants

Education	#	fraction
No answer	1	0.02
High School Diploma	22	0.37
Bachelor's Degree	11	0.19
Master's Degree	11	0.19
Doctorate's Degree	3	0.05
Other	11	0.9

Table 5.6: Country of participants

Country	#	fraction
Argentina	2	0.03
Bhutan	1	0.02
China	1	0.02
Estonia	2	0.03
France	4	0.07
Guyana	1	0.02
Hungary	1	0.02
India	3	0.05
Israel	1	0.02
Italy	3	0.05
Mexico	2	0.03
Netherlands	13	0.22
Philippines	1	0.02
Portugal	2	0.03
Spain	8	0.14
United Kingdom	2	0.03
United States	12	0.20

set, which supports the assumption that hackers do not commonly use multiple IP addresses. This statement does require further research, since this experiment allowed participants to compromise the system and therefore did not require them to hide their tracks.

The 59 participants performed a total of 12915 attacks, illustrated in Table 5.7. The system contained filename injection, SQL injection, cross site scripting, extraneous files, XSL(T) injection and XML external entity vulnerabilities. The table shows that many other attacks were also performed, since the participants did not know which vulnerabilities were present in the system.

Table 5.7: Hacker experiment: Signatures and matches

signature	matches	Checklist item
1120001	18	Check for default and predictable accounts
1120003	1	Check for default and predictable accounts
1210001	3616	Check for filename injection / path traversal
1210002	142	Check for filename injection / path traversal
1220001	592	Check for SQL injection
1220002	3750	Check for SQL injection
1220003	270	Check for SQL injection
1230001	1043	Check for cross site scripting
1230003	67	Check for cross site scripting
1230004	23	Check for cross site scripting
1230005	55	Check for cross site scripting
1230007	2	Check for cross site scripting
1240001	69	Check for system command injection
1310001	5	Check for uploading of (dynamic) scripts
20210001	2752	Check for extraneous files in document root
20340001	5	Check for accessible CVS/SVN directories
20340002	23	Check for accessible CVS/SVN directories
20370001	279	Check for accessible non-parsed dynamic scripts
20650001	115	Check for brute-force username enumeration
20760001	50	Check for XSL(T) injection
20770001	20	Check for SSI injection
20790001	5	Check for dynamic scripting injection
20810001	12	Check for XML external entity parsing

An import function of the TDS is the ability to assign the correct skill to threats. Using the questionnaire to determine the skill of the participant and the skill defined by the TDS, it is possible to calculate the correlation between the two. Using Spearman's Rank Order Correlation Coefficient, the correlation is calculated to be 0.598. In a data set of 59 participants, this is statistically significant (a correlation of 0.30 is required for the significance level of 1%).

To verify the results of the questionnaire and the results of the TDS, a sample

Table 5.8: Skill as identified by the questionnaire and the TDS

Skill by Questionnaire	Skill by TDS	Number
None	None	3
None	Basic	5
None	Advanced	1
Basic	None	1
Basic	Basic	18
Basic	Advanced	7
Advanced	none	1
Advanced	Basic	15
Advanced	Advanced	8

of 10 randomly selected participants is manually reviewed. During this manual review, the HTTP requests are analyzed and attacks are annotated. Then, using the same definition of skill as used in the questionnaire and the TDS, the skill of the participant is determined by the skill level of her most difficult attack. The results of this manual review are illustrated in Table 5.9. This table clearly shows that the skill of the participant is not adequately assessed using the questionnaire. The result of this experiment is therefore that the TDS can assess the skill of threats more accurately than a questionnaire could do. In a practical scenario, this means that the system will provide better results than would be possible if the threats of that system filled in a questionnaire.

Table 5.9: Skill determined by the questionnaire, manual review and the TDS

id	questionnaire skill	manual skill	TDS skill
1	none	basic	basic
2	advanced	advanced	advanced
3	basic	advanced	advanced
4	advanced	basic	basic
5	advanced	basic	basic
6	basic	basic	basic
7	basic	basic	basic
8	basic	basic	basic
9	basic	none	none
10	none	none	none

Finally, the android profiler can be evaluated using these results. The android profiler should identify each threat as being a human but failed to do this for 13 of the 59 participants (22%). The reason for this poor performance is the way the web application is set up; when the threat enters his or her attack in

the search field, the field is transmitted to the server after each character using XmlRpc-Requests. Each request is detected separately as an attack, and the timing between the requests is inhuman.

5.4 Discussion

During the design of the system, many assumptions are made that are not scientifically proven. These assumptions include:

- the simplification that each threat only uses one IP address
- the Certified Secure checklists accurately define basic and an advanced skills
- computer programs can be detected by the rate of their attacks

Therefore, the system is designed and implemented to be easily configurable to accommodate different opinions.

The assumption that each threat uses only one IP address was proven to be wrong for one threat in the real network experiment. Six threats were detected which are actually one that is camouflaging her attacks by using five computers. Instead of one threat performing 180 attacks (intensity class 3), the threat is now listed as six threats performing 30 attacks each (intensity class 2). In the hacker experiment, also one out of 59 threats used multiple (2) IP addresses to perform her attacks. While these results show that threats sometimes use multiple IP addresses, it is not a commonly used tactic and resulted in both cases in more threats being reported with a lower intensity class each.

The performance of the Threat Detection System is measured by the false positive rate, the false negative rate, and the accuracy of the profiles determined for each threat. The real network experiment was used to identify the false positive rate on a real network, but could not identify the false negative rate or the accuracy since the threats are inherently unknown on a real network. In this real network, one out of the 47 detected threats was a false positive. Internally, the Intrusion Detection System component detected 1343 attacks of which 540 were no attacks and therefore false positives. This high number of false positives would be a bad performance for any Intrusion Detection System, but since all false positives are triggered by the same threat, the performance of the Threat Detection System is only affected slightly. It could even be argued that this one threat is doing something unusual, and should therefore be investigated anyway.

The false negative rate of the Threat Detection System could not be identified using a real network, since the threats on this network are unknown. It would be useful if future research provided a data set containing real network traffic and a labeling of all threats and attacks in this data set. For this research,

an experiment is conducted inviting many security experts and hackers to compromise a website specifically designed to be compromised. Participants first register and are therefore known, but not to the system. In this experiment, the system detects threats in a data set containing mostly threats, which is only possible since the system uses a signature based intrusion detection system component and no machine learning or anomaly based components. At the end of the experiment, the system detected all threats resulting in a false negative rate of 0. This means that for each threat, at least one attack is detected. Since threats try many different attacks on a website, trying to compromise it, the system performs well while only detecting 35% of all possible web attacks. A problem with this experiment is that the vulnerabilities which were introduced are all covered by the signature set. This allows for the argument that a website containing different vulnerabilities, which are not part of the signature set, could be compromised without the Threat Detection System noticing it. While this is true, the participants of the experiment were not told which vulnerabilities were present in the website and they could try whatever attack they wanted to perform; all 59 participants chose to perform at least one attack that was detected by the system.

Finally, the accuracy of the profiles determined for each threat is identified. The participants in the experiment filled out a questionnaire after performing their attacks on the website. This questionnaire was supposed to identify the skill of each participant which could then be compared to the skill identified by the Threat Detection System. The correlation between the skill determined by the questionnaire and the skill determined by the TDS is 0.598, which is significant for a data set of 59 (a correlation of 0.30 is required for the significance level of 1%). However, such a number is vague and does not show the whole picture. When manually examining the network traffic of 10 randomly selected participants, it turns out that the participants cannot accurately answer questions about the attacks they performed. When manually reviewing their network traffic, labeling their attacks and manually identifying their skill class based on the criteria described in section 4.4, the TDS was identifying the correct skill class for each of the 10 participants. This means that the TDS can more accurately identify the skill class of a threat, than would be possible using the questionnaire. While this could be a fault in the design of the questionnaire, the fact that the TDS identified the skill identically to a manual review by a security expert, proves the skill class identified for each threat to be accurate.

The android class determined for each threat is less accurate than the skill class, as 22% of the human participants are determined to be a computer. This inaccuracy in inherent to determining such an abstract property. However, the android class is useable for security experts as long as the error rate is taken into account.

Chapter 6

Conclusion

The system presented in this thesis automatically detects and profiles threats on a real network. When placed in a new network without any changes to its configuration, one of the 47 detected threats was falsely identified as a threat (a false positive) and two of the 47 detected threats were falsely identified as being a human. When placed in a network containing only human threats, the system correctly identifies the skill of each threat and identifies them as being human 78% out the time. To achieve this, three contributions to the professional and academic world are made.

First, the signatures for an open source intrusion detection system (Sort [59]) are developed to detect many generic types of attacks (like SQL Injection and Cross Site Scripting). Currently, signatures used in such systems can detect only specific attack instances to known vulnerabilities. The presented set of signatures allows intrusion detection systems to detect attacks on unknown vulnerabilities.

Second, an extension on this intrusion detection system using the new set of signatures is developed shifting the focus from the attacks detected on the network to the actual threats (a human or autonomous computer program) attacking the network and their properties (such as skill). This paradigm shift is a new approach to intrusion detection, and the system is therefore named a Threat Detection System (TDS). By focussing on the threats instead of the attacks, the system is not required to achieve a 100% detection rate of attacks on the network. Threats are detected when they perform at least few attacks which are detected by the system.

Finally, algorithms are presented to determine three properties for each threat: skill, intensity of the attacks and whether the threat is a human or an autonomous computer program. These algorithms are accompanied by guidance on how to interpret their results and how to apply them to risk management and network security monitoring. This allows experts to directly use this system without changes to the configuration.

On itself, the presented signatures for the open source intrusion detection system allow academics to research real threats on real networks and allow security experts to monitor actual attacks on their computer networks. The complete Threat Detection System allows security experts evaluating the risk of a vulnerability to retrieve the actual number of threats that are active on the network and are capable of exploiting this vulnerability. It also allows security experts monitoring the security of a network to focus only on threats that are capable of compromising the network. Academics can use this system as basis for their own research on threats and adapt it to detect different properties or automatically respond to certain types of threats.

Combined, these contributions improve the accuracy of risk management and the efficiency of network security monitoring. This meets the problem definition for this thesis: "Design a system to autonomously detect and profile threats on a computer network to improve risk management and network security monitoring."

6.1 Recommendations for further research

This research covers many aspects of computer security and therefore hasn't covered every detail in depth. A number of topics could benefit from further research.

- One of the main assumptions of this thesis is that security experts cannot correctly define the likelihood of a vulnerability being exploited. It also assumes that security experts can correctly identify the threat profile that is required to exploit a vulnerability. These assumptions have not been verified in this research and could benefit from further research.
- In this research, attack types that could not be implemented without triggering many false positives are not implemented at all. Using a different Intrusion Detection System that has different capabilities in defining the signatures could help creating a complete set.
- The signatures presented in this research focus only on web attacks. Further research could implement signatures that match generic attacks aimed at different services or at different layers in the network stack.
- The lack of a publicly available data set containing labelled attacks and threats prevents research as presented in this thesis to compare its performance to other research. Further research could really improve the field of intrusion detection by presenting an accurate, realistic and labelled data set containing both attacks and normal network traffic.
- Further research of threat classification could better classify the attacks in threats, possible using more features than just the source IP address.

One camouflaging attack was detected during the test on a live network, so this is happening.

- This research uses three threat properties to define a threat profile. Further research could try to create more properties and perhaps use external databases (netflow information, p0f information, low level packet data) to assign different and more accurate properties to threats.

Appendices

Appendix A

Attack and Penetration test documentation

This appendix contains the sample documentation of a SQL Injection vulnerability by the security company Pine Digital Security. This documentation is included and translated (with permission) from the Pine Digital Security Sample Attack and Penetration Test report [58].

A.1 Description

When the communication with a SQL database includes user input, there is a possibility for SQL Injection. This problem allows a threat to have his input interpreted as SQL commands (instead of just 'data').

A.2 Result

Result	Vulnerability Score
Fail	6.8

A SQL Injection vulnerability is found in the order history of the Test Product. Figure A.1 illustrates the normal functionality to query order information. When the order number is changed from 6 to 6 AND 1=1, a different page is displayed is illustrated in Figure A.2.

The user input (6 AND 1=2) is internally used in a SQL command. The part AND 1=2 causes no order information to be found. When this input is changed to AND 1=1, the order information can be found as illustrated in Figure A.3.

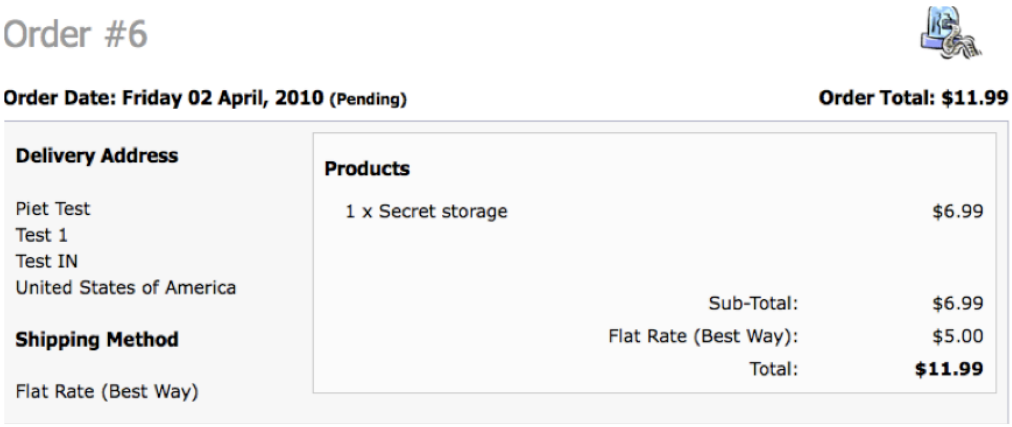

Figure A.1: http://oscommerce.hackme.certifiedsecure.com/account.php?orders=6

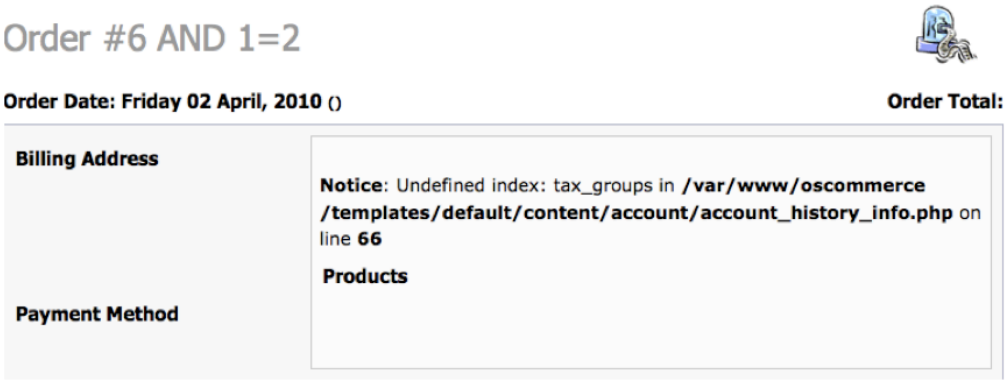

Figure A.2: http://oscommerce.hackme.certifiedsecure.com/account.php?orders=6 AND 1=2

This vulnerability allows threats to alter the SQL commands. This allows them to retrieve information from the database, like credit-card information as illustrated in Figure A.4.

A.3 Vulnerability Score

The vulnerability score is based on the following categories and is calculated using the "Common Vulnerability Scoring System" version 2[25].

Access Vector: Network The vulnerability can be exploited from anywhere on the Internet.

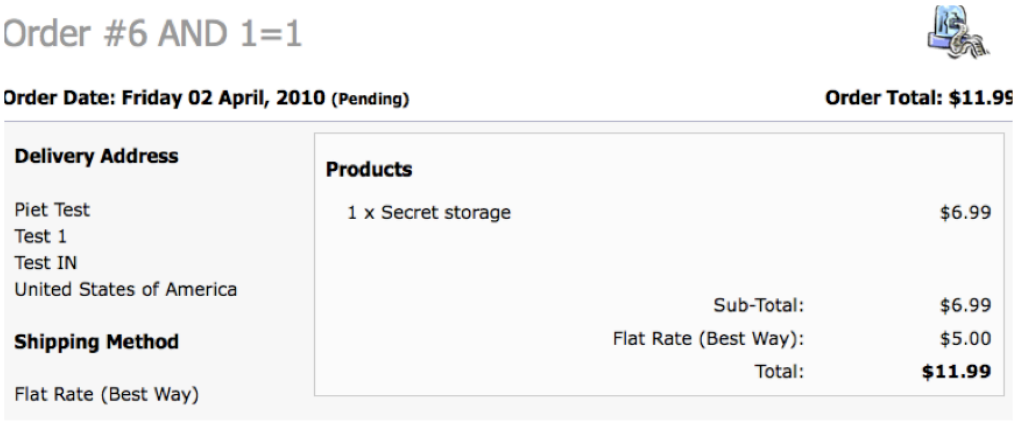

Figure A.3: http://oscommerce.hackme.certifiedsecure.com/account.php?orders=6 AND 1=1

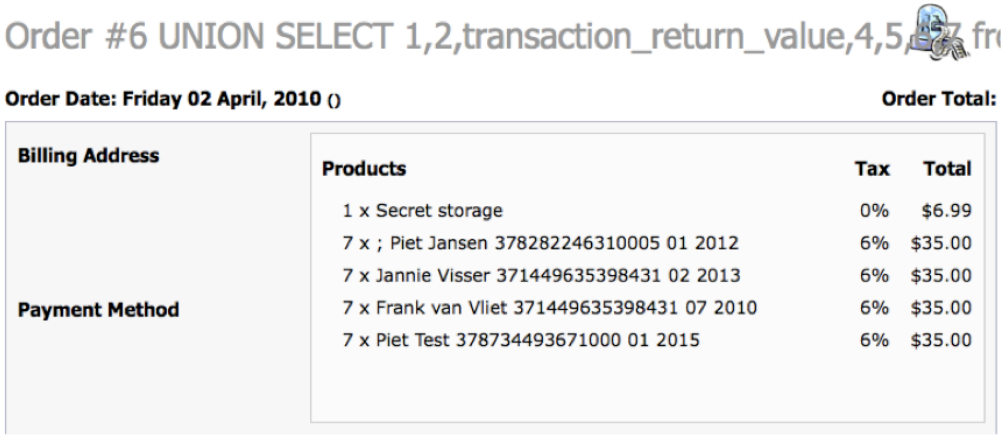

Figure A.4: http://oscommerce.hackme.certifiedsecure.com/account.php?orders=6 UNION SELECT 1,2,transaction_return_value,4,5,6,7 from osc_orders_transactions_history #

Access Complexity: Low The vulnerability can be exploited at any moment, without restrictions.

Authentication: Single A single authentication is required to exploit this vulnerability and anyone can get an account for free via a simple registration on the site.

Confidentiality: Complete All information stored in the system can be compromised using this vulnerability.

Integrity: None This vulnerability has no direct impact on the integrity of the system.

Availability: None This vulnerability has no direct impact on the availability of the system.

A.4 Solution

It is important to check the source code of the Test Product for any user input that is used in SQL commands without validation or escaping (for example using the PHP mysql_real_escape_string() function). More information about this function can be found at http://php.net/manual/en/function.mysql-real-escape-string.php.

Appendix B

Snort configuration

```
# Snort configuration file for the Threat Detection System (TDS
    )
#
# by Frank van Vliet
#    frank@certifiedsecure.nl


# The IDS must detect attacks originating from any network
# (EXTERNAL_NET) and destined to any network (HOME_NET).
var EXTERNAL_NET any
var HOME_NET any

# Each server in the destination network (HOME_NET) can be
# an HTTP server (HTTP_SERVERS) listening on port 80
# (HTTP_PORTS).
var HTTP_SERVERS $HOME_NET
portvar HTTP_PORTS [80]

# Calculate the checksum for IP, TCP packets. This has a
# slight impact on the performance, but does allow the IDS
# to more accurately reassemble IP packets and TCP streams.
# The UDP and ICMP protocols are outside the requirements
# and their checksum is thus ignored.
config checksum_mode: none

# Disable alerts that are generated when decoding the
# packets. This will result is some false negatives when
# threats are trying to evade the IDS, but turning it on
# will result in many false positives caused by network
# errors and client software problems.
config disable_decode_alerts
config disable_ipopt_alerts
config disable_tcpopt_alerts
config disable_tcpopt_experimental_alerts
config disable_tcpopt_obsolete_alerts
```

```
config disable_tcpopt_ttcp_alerts
config disable_ttcp_alerts

# The Frag3 preprocessor will reassemble IP packets. A
# specific problem with IP reassembly is the the Ptacok
# & Newsham IDS evasion technique:
# http://www.snort.org/docs/idspaper/
#
# The preprocessor is configured to track up to 65536
# fragments, which is the maximum number of fragments
# of one IP packet.
#
# The preprocessor is futhermore configured to wait for
# 180 seconds for before they timeout.
#
# This should be adequate for normal traffic. Later
# versions of this TDS should focus on improving this
# configuration to prevent IDS evasion, for example
# by automatically detecting the fragmentation
# properties of each host on the network.
preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: timeout 180

# The Stream5 preprocessor will reassemble TCP flows.
# Like Frag3, this reassembly can be evaded by attacks.
#
# The preprocessor is configured to track up to
# 256000 tcp connections simultaniously, to track TCP
# connections and not track UDP or ICMP connections
# (since UDP and ICMP attacks are outside the
# requirements of this TDS).
#
# The preprocessor is configured to only reassemble
# connections on 80.
preprocessor stream5_global: max_tcp 256000, \
                             track_tcp yes, \
                             track_udp no, \
                             track_icmp no
preprocessor stream5_tcp: ports both 80

# The http_inspect preprocessor will decode all HTTP
# traffic.
#
# The preprocessor is configured to use the default
# unicode codepage 1252 (used for decoding Unicode
# characters). It is possible that different
# webservers use different codepages, which would
# allow threats to evade the IDS. Future versions of
# this TDS could try to solve this problem.
#
# The preprocessor is configured to use the default
# profile for each HTTP server. This profile uses
```

```
# the common methods of decoding, that could be
# different from the actual HTTP servers. The
# preprocessor should only decode traffic on
# port 80, and decode up to 1000 characters of
# the client request. The alerts generated while
# decoding the HTTP traffic are ignored since they
# contain too many false positives. Future versions
# of this IDS could try to improve this situation.
preprocessor http_inspect: global iis_unicode_map \
                                    unicode.map 1252
preprocessor http_inspect_server: \
    server default \
    profile all \
    ports { 80 } \
    client_flow_depth 1000 \
    post_depth 500 \
    no_alerts

# Unified logfiles are stored in binary format and
# contain both the alert information and the packet
# contents.
output alert_unified: filename snort.alert
output log_unified: filename snort.log

# The following signatures (named rules in Snort)
# are loaded:
#   * basic-web.rules: Rules matching attacks from
#     the CS Basic Web Application Scan Checklist,
#   * advanced-web.rules: Rules matching attacks
#     from the CS Advanced Web Application Scan
#     Checklist.
var RULE_PATH rules
include $RULE_PATH/basic-web.rules
include $RULE_PATH/advanced-web.rules
```

Listing B.1: Configuration for the Snort Intrusion Detection System

Appendix C

Checklist comparison

Certified Secure Basic Web Application Scan Checklist			
Id	Description	OWASP Top 10	Sans Top 25
1.0	**Authentication and Authorization**		
1.1	Check for client side authentication	3	6
1.2	Check for default and predictable accounts	3, 6	6
1.3	Check for identifier based authorization	4	5
2.0	**User Input**		
2.1	Check for filename injection / path traversal	1	7
2.2	Check for SQL injection	1	2
2.3	Check for cross site scripting	2	1
2.4	Check for system command injection	1	9
3.0	**File Upload**		
3.1	Check for uploading of (dynamic) scripts		8
4.0	**Sessions**		
4.1	Check for Cross Site Request Forgery	5	4

Certified Secure Advanced Web Application Scan Checklist			
Id	Description	OWASP Top 10	Sans Top 25
1.0	**Multi-system Services**		
1.1	Check for HTTP request smuggling		
2.0	**Design**		
2.1	Check for extraneous files in document root		
3.0	**Information Disclosure**		
3.1	Check for too verbose error messages	6	16

Certified Secure Advanced Web Application Scan Checklist			
Id	Description	OWASP Top 10	Sans Top 25
3.2	Check for debug enabling using a predictable parameter		
3.3	Check for valuable information in robots.txt		
3.4	Check for accessible CVS/SVN directories		
3.5	Check for accessible configuration directories		
3.6	Check for accessible backup files		
3.7	Check for accessible non-parsed dynamic scripts		
4.0	**Privacy and Confidentiality**		
4.1	Check for missing anti-caching headers		
4.2	Check for unencrypted transmissions of sensitive information	9	10
4.3	Check for sensitive information stored in cookies		
4.4	Check for sensitive information in externally archived pages		
5.0	**Integrity**		
5.1	Check for client side state management		
6.0	**Authentication and Authorization**		
6.1	Check for missing authentication	8	19
6.2	Check for authentication based on the knowledge of a secret URL		
6.3	Check for identifier based authentication		
6.4	Check for too verbose authentication-failure logging		
6.5	Check for brute-force username enumeration		
6.6	Check for brute-force password guessing		
6.7	Check for denial of service by locking out accounts		
6.8	Check for authentication or authorization based on obscurity		
7.0	**User Input**		
7.1	Check for double decoding of headers / parameters		
7.2	Check for XML injection	1	
7.3	Check for XPath injection	1	
7.4	Check for LDAP injection	1	
7.5	Check for HTTP header injection	1	
7.6	Check for XSL(T) injection	1	

Certified Secure Advanced Web Application Scan Checklist			
Id	Description	OWASP Top 10	Sans Top 25
7.7	Check for SSI injection	1	
7.8	Check for resource identifier injection	10	23
7.9	Check for dynamic scripting injection	1	13
7.10	Check for regular expression injection	1	
8.0	**XML**		
8.1	Check for XML external entity parsing		
8.2	Check for XML external DTD parsing		
9.0	**File Upload**		
9.1	Check for uploading outside of intended directory		
9.2	Check for incorrect handling of very large files		
9.3	Check for local file disclosure via upload filename		
9.4	Check for uploading of configuration files		
10.0	**Email**		
10.1	Check for automated spamming via (feedback) scripts		
11.0	**Sessions**		
11.1	Check for session-cookies without the secure flag		
11.2	Check for session-cookies without the httponly flag		
11.3	Check for predictable session-ids		
11.4	Check for session collisions		
11.5	Check for session-fixation	3	
11.6	Check for external session-hijacking		
11.7	Check for insecure transmission of session-cookies	3	
11.8	Check for missing session revocation if session-id transmitted unencrypted		
12.0	**Cryptography**		
12.1	Check for unproven cryptographic algorithms		24

Appendix D

First version of the signatures for the Basic Web Application Scan Checklist

```
# Rules for the Certified Secure Basic Web Application Scan
    Checklist
# 1.0 Authentication and Authorization
## 1.1 Check for client side authentication or authorization
### This cannot be easily captured in a rule
## 1.2 Check for default and predictable accounts
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "name=admin"; nocase; pcre:"/pass*=admin/UPi"; sid
    :1120001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "user=admin"; nocase; pcre:"/pass*=admin/UPi"; sid
    :1120002;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "name=admin"; nocase; pcre:"/pass*=password/UPi"; sid
    :1120003;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "user=admin"; nocase; pcre:"/pass*=password/UPi"; sid
    :1120004;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
```

```
    "name=guest"; nocase; pcre:"/pass*=guest/UPi"; sid
    :1120005;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "user=guest"; nocase; pcre:"/pass*=guest/UPi"; sid
    :1120006;)
## 1.3 Check for identifier based authorization
### This cannot be easily captured in a rule

# 2.0 User Input
## 2.1 Check for filename injection / path traversal
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.1 Check for filename
    injection / path traversal"; flow:to_server,established;
    content:"="; content:"../"; distance:0; content:"../";
    distance:0; sid:1210001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.1 Check for filename
    injection / path traversal"; flow:to_server,established;
    content:"="; content:"..\\"; distance:0; content:"..\\";
    distance:0;sid:1210002;)

## 2.2 Check for SQL injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.2 Check for SQL injection";
     flow:to_server,established; pcre:"/OR\s+'?\d+'?\s*=\s*'?\d
    +/UPi"; sid:1220001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.2 Check for SQL injection";
     flow:to_server,established; pcre:"/AND\s+'?\d+'?\s*=\s*'?\
    d+/UPi"; sid:1220002;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.2 Check for SQL injection";
     flow:to_server,established; pcre:"/UNION[\s(]+SELECT/UPi";
     sid:1220003;)

## 2.3 Check for cross site scripting
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"</
    script>"; nocase; sid:1230001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"<script
     src"; nocase; sid:1230002;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"
    javascript:alert"; nocase; sid:1230003;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
```

```
    scripting"; flow:to_server,established; uricontent:"<iframe
    "; nocase; sid:1230004;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"</script>"
    ; http_client_body; nocase; sid:1230005;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"<script
    src"; http_client_body; nocase; sid:1230006;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"javascript
    :alert"; http_client_body; nocase; sid:1230007;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"<iframe";
    http_client_body; nocase; sid:1230008;)

## 2.4 Check for system command injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.4 Check for system command
    injection"; flow:to_server,established; pcre:"/[|`]\s*(
    sleep|ping|id|ls|dir|whoami|reboot|shutdown|nc|socat)
    *[|`]/UPi"; sid:1240001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.4 Check for system command
    injection"; flow:to_server,established; pcre:"/\;\s*(sleep|
    ping|id|ls|dir|whoami|reboot|shutdown|nc|socat) /UPi"; sid
    :1240002;)

# 3.0 File Upload
## 3.1 Check for uploading of (dynamic) scripts
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 3.1 Check for uploading of (
    dynamic) scripts"; flow:to_server,established; pcre:"/
    filename=.*\.(php|cgi|shtml|asp|aspx|jsp)/i";content:"POST"
    ; http_method; nocase; sid:1310001;)

# 4.0 Cross Site Request Forgery
## This cannot be easily captured in a rule
```

Listing D.1: First version of the signatures for the Basic Web Application Scan Checklist

Appendix E

First version of the signatures for the Advanced Web Application Scan Checklist

```
# Rules for the Certified Secure Advanced Web Application Scan
    Checklist

#1.0    Multi-system Services
##1.1   Check for HTTP request smuggling

#2.0    Design
##2.1   Check for extraneous files in document root
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 2.1 Check for extraneous
    files in document root"; flow:to_server,established; pcre:"
    /\/(.psql_history|acceso\/|access-log|access.log|access\/|
    access_log|(abbreviated in this illustration)|phpinfo.php)$
    /Ui"; detection_filter: track by_src, count 10, seconds 60;
     sid:20210001;)
#3.0    Information Disclosure
##3.1   Check for too verbose error messages
##3.2   Check for debug enabling using a predictable parameter
##3.3   Check for valuable information in robots.txt
##3.4   Check for accessible CVS/SVN directories
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 3.4 Check for accessible
    CVS/SVN directories"; flow:to_server,established;
    uricontent:"/.svn/"; sid:20340001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 3.4 Check for accessible
    CVS/SVN directories"; flow:to_server,established;
    uricontent:"/CVS/"; sid:20340002;)
##3.5   Check for accessible configuration directories
### Handled by 2.1
```

```
##3.6   Check for accessible backup files
### Handled by 2.1
##3.7   Check for accessible non-parsed dynamic scripts
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 3.7 Check for accessible
    non-parsed dynamic scripts"; flow:to_server,established;
    pcre:"/(\.\$\$\$|\.\$db|\.113|\.abbu|\.abk|\.bac|\.bak|\.
    bck|\.bcm|\.bdb|\.bkp|\.bks|\.bps|\.bup|\.cbk|\.da0|\.dbk
    |\.dov|\.gho|\.jbk|\.llx|\.mem|\.nb7|\.nbk|\.nco|\.nrs|\.
    oeb|\.old|\.oyx|\.qbx|\.qic|\.tbk|\.tmp|\.win|\.win|\.xlk
    |\.orig|\.swp|\.swo|~|\.backup|\.v)$/Ui"; sid:20370001;)
#4.0    Privacy and Confidentiality
##4.1   Check for missing anti-caching headers
##4.2   Check for unencrypted transmissions of sensitive
    information
##4.3   Check for sensitive information stored in cookies
##4.4   Check for sensitive information in externally archived
    pages
#5.0    Integrity
##5.1   Check for client side state management
#6.0    Authentication and Authorization
##6.1   Check for missing authentication
##6.2   Check for authentication based on knowledge of a secret
     URL
##6.3   Check for identifier based authentication
##6.4   Check for too verbose authentication-failure logging
##6.5   Check for brute-force username enumeration
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-application-Scan 6.5 Check for brute-force
    username enumeration"; flow:to_server,established; content:
    "name="; nocase; pcre:"/pass*=/UPi"; detection_filter:
    track by_src, count 20, seconds 60; sid:20650001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 6.5 Check for brute-force
    username enumeration"; flow:to_server,established; content:
    "user="; nocase; pcre:"/pass*=/UPi"; detection_filter:
    track by_src, count 20, seconds 60; sid:20650002;)
##6.6   Check for brute-force password guessing
# by 6.5
##6.7   Check for denial of service by locking out accounts
# by 6.5
##6.8   Check for authentication or authorization based on
    obscurity
##6.9   Check for changing a password without knowledge of the
    old password
#7.0    User Input
##7.1   Check for double decoding of headers / parameters
# it is not possible to match this rule without many many false
     positives
##7.2   Check for XML injection
# it is not possible to match this rule without many many false
     positives
```

```
##7.3   Check for XPath injection
# it is not possible to match this rule without many many false
     positives
##7.4   Check for LDAP injection
# it is not possible to match this rule without many many false
     positives
##7.5   Check for HTTP header injection
##7.6   Check for XSL(T) injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 7.6 Check for XSL(T)
    injection"; flow:to_server,established; pcre:"/<xsl:value-
    of /UPi"; sid:20760001;)
##7.7   Check for SSI injection
##7.8   Check for resource identifier injection
# it is not possible to match this rule without many many false
     positives
##7.9   Check for dynamic scripting injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 7.9 Check for dynamic
    scripting injection"; flow:to_server,established; pcre:"/(
    include|passthru|exec|system|)\(*\$_(GET|POST|COOKIE)\[/UPi
    "; sid:20790001;)
##7.10  Check for regular expression injection
#8.0    XML
##8.1   Check for XML external entity parsing
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 8.1 Check for XML external
     entity parsing"; flow:to_server,established; pcre:"/<\!
    ENTITY\s+\sSYSTEM\s/UPi"; sid:20810001;)
##8.2   Check for XML external DTD parsing
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 8.2 Check for XML external
     DTD parsing"; flow:to_server,established; pcre:"/<\!
    DOCTYPE\s+\sSYSTEM\s/UPi"; sid:20820001;)
#9.0    File Upload
##9.1   Check for uploading outside of intended directory
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.1 Check for uploading
    outside of intended directory"; flow:to_server,established;
     pcre:"/filename=.*\.\./i";content:"POST"; http_method;
    nocase; sid:2091001;)
##9.2   Check for incorrect handling of very large files
##9.3   Check for local file disclosure via upload filename
##9.4   Check for uploading of configuration files
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.4 Check for uploading of
     configuration files"; flow:to_server,established; pcre:"/
    filename=.*\.htaccess/i";content:"POST"; http_method;
    nocase; sid:2094001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.4 Check for uploading of
     configuration files"; flow:to_server,established; pcre:"/
```

```
    filename=.*\.htpasswd/i";content:"POST"; http_method;
    nocase; sid:2094002;)
#10.0   Email
##10.1  Check for automated spamming via (feedback) scripts
#11.0   Sessions
##11.1  Check for session-cookies without the secure flag
##11.2  Check for session-cookies without the httponly flag
##11.3  Check for predictable session-ids
##11.4  Check for session collisions
##11.5  Check for session-fixation
##11.6  Check for external session-hijacking
##11.7  Check for insecure transmission of session-cookies
##11.8  Check for missing session revocation on unencrypted
    session-id transmission
#12.0   Cryptography
##12.1  Check for unproven cryptographic algorithms
```

Listing E.1: First version of the signatures for the Advanced Web Application Scan Checklist

Appendix F

Final version of the signatures for the basic web application scan checklist

```
# Rules for the Certified Secure Basic Web Application Scan
    Checklist

# 1.0 Authentication and Authorization
## 1.1 Check for client side authentication or authorization
### This cannot be easily captured in a rule
## 1.2 Check for default and predictable accounts
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "name=admin"; nocase; pcre:"/pass*=admin/UPi"; sid
    :1120001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "user=admin"; nocase; pcre:"/pass*=admin/UPi"; sid
    :1120002;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "name=admin"; nocase; pcre:"/pass*=password/UPi"; sid
    :1120003;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "user=admin"; nocase; pcre:"/pass*=password/UPi"; sid
    :1120004;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
```

```
    "name=guest"; nocase; pcre:"/pass*=guest/UPi"; sid
    :1120005;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 1.2 Check for default and
    predictable accounts"; flow:to_server,established; content:
    "user=guest"; nocase; pcre:"/pass*=guest/UPi"; sid
    :1120006;)
## 1.3 Check for identifier based authorization
### This cannot be easily captured in a rule


# 2.0 User Input
## 2.1 Check for filename injection / path traversal
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.1 Check for filename
    injection / path traversal"; flow:to_server,established;
    content:"="; content:"../"; distance:0; content:"../";
    distance:0; sid:1210001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.1 Check for filename
    injection / path traversal"; flow:to_server,established;
    content:"="; content:"..\\"; distance:0; content:"..\\";
    distance:0;sid:1210002;)

## 2.2 Check for SQL injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.2 Check for SQL injection";
     flow:to_server,established; pcre:"/OR\s+'?\d+'?\s*=\s*'?\d
    +/UPi"; sid:1220001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.2 Check for SQL injection";
     flow:to_server,established; pcre:"/AND\s+'?\d+'?\s*=\s*'?\
    d+/UPi"; sid:1220002;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.2 Check for SQL injection";
     flow:to_server,established; pcre:"/UNION[\s(]+SELECT/UPi";
     sid:1220003;)

## 2.3 Check for cross site scripting
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"</
    script>"; nocase; sid:1230001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"<script
     src"; nocase; sid:1230002;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"
    javascript:alert"; nocase; sid:1230003;)
```

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; uricontent:"<iframe
    "; nocase; sid:1230004;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"</script>"
    ; http_client_body; nocase; sid:1230005;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"<script
    src"; http_client_body; nocase; sid:1230006;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"javascript
    :alert"; http_client_body; nocase; sid:1230007;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.3 Check for cross site
    scripting"; flow:to_server,established; content:"<iframe";
    http_client_body; nocase; sid:1230008;)

## 2.4 Check for system command injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.4 Check for system command
    injection"; flow:to_server,established; pcre:"/[|`]\s*(
    sleep|ping|id|ls|dir|whoami|reboot|shutdown|nc|socat)
    *[|`]/UPi"; sid:1240001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 2.4 Check for system command
    injection"; flow:to_server,established; pcre:"/\;\s*(sleep|
    ping|id|ls|dir|whoami|reboot|shutdown|nc|socat) /UPi"; sid
    :1240002;)

# 3.0 File Upload
## 3.1 Check for uploading of (dynamic) scripts
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Basic-Web-Application-Scan 3.1 Check for uploading of (
    dynamic) scripts"; flow:to_server,established; pcre:"/
    filename=.*\.(php|cgi|shtml|asp|aspx|jsp)/i";content:"POST"
    ; http_method; nocase; sid:1310001;)

# 4.0 Cross Site Request Forgery
## This cannot be easily captured in a rule

# 5.0 Miscellaneous
## 5.1 Check for application or setup specific problems
```

Listing F.1: Final version of the signatures for the Basic Web Application Scan Checklist

Appendix G

Final version of the signatures for the advanced web application scan checklist

```
# Rules for the Certified Secure Advanced Web Application Scan
    Checklist

#1.0    Multi-system Services
##1.1   Check for HTTP request smuggling

#2.0    Design

##2.1   Check for extraneous files in document root
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 2.1 Check for extraneous
    files in document root"; flow:to_server,established; pcre:"
    /\/(.psql_history|acceso\/|access-log|access.log|access\/|
    access_log|(abbreviated in this illustration)|phpinfo.php)$
    /Ui"; detection_filter: track by_src, count 100, seconds
    60; sid:20210001;)
#3.0    Information Disclosure
##3.1   Check for too verbose error messages
##3.2   Check for debug enabling using a predictable parameter
##3.3   Check for valuable information in robots.txt
##3.4   Check for accessible CVS/SVN directories
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 3.4 Check for accessible
    CVS/SVN directories"; flow:to_server,established;
    uricontent:"/.svn/"; sid:20340001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 3.4 Check for accessible
    CVS/SVN directories"; flow:to_server,established;
    uricontent:"/CVS/"; sid:20340002;)
##3.5   Check for accessible configuration directories
```

```
### Handled by 2.1
##3.6   Check for accessible backup files
### Handled by 2.1
##3.7   Check for accessible non-parsed dynamic scripts
#alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg
    :"CS-Advanced-Web-Application-Scan 3.7 Check for accessible
     non-parsed dynamic scripts"; flow:to_server,established;
    pcre:"/(\.\$\$\$/\.\$db/\.113/\.abbu/\.abk/\.bac/\.bak/\.
    bck/\.bcm/\.bdb/\.bkp/\.bks/\.bps/\.bup/\.cbk/\.da0/\.dbk
    /\.dov/\.gho/\.jbk/\.llx/\.mem/\.nb7/\.nbk/\.nco/\.nrs/\.
    oeb/\.old/\.oyx/\.qbx/\.qic/\.tbk/\.tmp/\.win/\.win/\.xlk
    /\.orig/\.swp/\.swo/~/\.backup/\.v)$/Ui"; sid:20370001;)

#4.0    Privacy and Confidentiality
##4.1   Check for missing anti-caching headers
##4.2   Check for unencrypted transmissions of sensitive
    information
##4.3   Check for sensitive information stored in cookies
##4.4   Check for sensitive information in externally archived
    pages
#5.0    Integrity
##5.1   Check for client side state management
#6.0    Authentication and Authorization
##6.1   Check for missing authentication
##6.2   Check for authentication based on knowledge of a secret
     URL
##6.3   Check for identifier based authentication
##6.4   Check for too verbose authentication-failure logging
##6.5   Check for brute-force username enumeration
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 6.5 Check for brute-force
    username enumeration"; flow:to_server,established; content:
    "name="; nocase; pcre:"/pass*=/UPi"; detection_filter:
    track by_src, count 20, seconds 60; sid:20650001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 6.5 Check for brute-force
    username enumeration"; flow:to_server,established; content:
    "user="; nocase; pcre:"/pass*=/UPi"; detection_filter:
    track by_src, count 20, seconds 60; sid:20650002;)
##6.6   Check for brute-force password guessing
# by 6.5
##6.7   Check for denial of service by locking out accounts
# by 6.5
##6.8   Check for authentication or authorization based on
    obscurity
##6.9   Check for changing a password without knowledge of the
    old password
#7.0    User Input
##7.1   Check for double decoding of headers / parameters
#alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg
    :"CS-Advanced-Web-Application-Scan 7.1 Check for double
    decoding of headers / parameters"; flow:to_server,
```

```
    established; content:"%2522"; sid:20710001;)
#alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg
    :"CS-Advanced-Web-Application-Scan 7.1 Check for double
    decoding of headers / parameters"; flow:to_server,
    established; pcre:"/^GET .*%25(22|27|3c|3e)/Bi"; sid
    :20710003;)
#alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg
    :"CS-Advanced-Web-Application-Scan 7.1 Check for double
    decoding of headers / parameters"; flow:to_server,
    established; content:"%253c"; nocase; sid:20710004;)
#alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg
    :"CS-Advanced-Web-Application-Scan 7.1 Check for double
    decoding of headers / parameters"; flow:to_server,
    established; content:"%253e"; nocase; sid:20710005;)
##7.2   Check for XML injection
# it is not possible to match this rule without many many false
     positives
##7.3   Check for XPath injection
# it is not possible to match this rule without many many false
     positives
##7.4   Check for LDAP injection
# it is not possible to match this rule without many many false
     positives
##7.5   Check for HTTP header injection
#alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg
    :"CS-Advanced-Web-Application-Scan 7.5 Check for HTTP
    header injection"; flow:to_server,established; pcre:"/\n+:
    +/U"; sid:20750005;)
##7.6   Check for XSL(T) injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 7.6 Check for XSL(T)
    injection"; flow:to_server,established; pcre:"/<xsl:value-
    of /UPi"; sid:20760001;)
##7.7   Check for SSI injection
##7.8   Check for resource identifier injection
# it is not possible to match this rule without many many false
     positives
##7.9   Check for dynamic scripting injection
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 7.9 Check for dynamic
    scripting injection"; flow:to_server,established; pcre:"/(
    include|passthru|exec|system|)\(*\$_(GET|POST|COOKIE)\[/UPi
    "; sid:20790001;)
##7.10  Check for regular expression injection
#8.0    XML
##8.1   Check for XML external entity parsing
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 8.1 Check for XML external
     entity parsing"; flow:to_server,established; pcre:"/<\!
    ENTITY\s+\sSYSTEM\s/UPi"; sid:20810001;)
##8.2   Check for XML external DTD parsing
```

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 8.2 Check for XML external
     DTD parsing"; flow:to_server,established; pcre:"/<\!
    DOCTYPE\s+\sSYSTEM\s/UPi"; sid:20820001;)
#9.0    File Upload
##9.1   Check for uploading outside of intended directory
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.1 Check for uploading
    outside of intended directory"; flow:to_server,established;
     pcre:"/filename=.*\.\.\//i";content:"POST"; http_method;
    nocase; sid:2091001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.1 Check for uploading
    outside of intended directory"; flow:to_server,established;
     pcre:"/filename=.*\.\.\\/i";content:"POST"; http_method;
    nocase; sid:2091002;)
##9.2   Check for incorrect handling of very large files
##9.3   Check for local file disclosure via upload filename
##9.4   Check for uploading of configuration files
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.4 Check for uploading of
     configuration files"; flow:to_server,established; pcre:"/
    filename=.*\.htaccess/i";content:"POST"; http_method;
    nocase; sid:2094001;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"
    CS-Advanced-Web-Application-Scan 9.4 Check for uploading of
     configuration files"; flow:to_server,established; pcre:"/
    filename=.*\.htpasswd/i";content:"POST"; http_method;
    nocase; sid:2094002;)
#10.0   Email
##10.1  Check for automated spamming via (feedback) scripts
#11.0   Sessions
##11.1  Check for session-cookies without the secure flag
##11.2  Check for session-cookies without the httponly flag
##11.3  Check for predictable session-ids
##11.4  Check for session collisions
##11.5  Check for session-fixation
##11.6  Check for external session-hijacking
##11.7  Check for insecure transmission of session-cookies
##11.8  Check for missing session revocation on unencrypted
    session-id transmission
#12.0   Cryptography
##12.1  Check for unproven cryptographic algorithms
```

Listing G.1: Final version of the signatures for the Advanced Web Application Scan Checklist

General References

[1] Y. Bai and H. Kobayashi. Intrusion detection systems: technology and development. In *Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on*, pages 710 – 715, March 2003.

[2] Richard Bejtlich. *The tao of network security monitoring, Beyond Intrusion Detection*. Addison Wesley, July 2007.

[3] Salem Benferhat, Tayeb Kenaza, and Aicha Mokhtari. A naive bayes approach for detecting coordinated attacks. *Computer Software and Applications Conference, Annual International*, 0:704–709, 2008.

[4] Damiano Bolzoni, Bruno Crispo, and Sandro Etalle. Atlantides: an architecture for alert verification in network intrusion detection systems. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–12, Berkeley, CA, USA, 2007. USENIX Association.

[5] Damiano Bolzoni and Sandro Etalle. Boosting web intrusion detection systems by inferring positive signatures. In *OTM '08: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*, pages 938–955, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] Damiano Bolzoni, Sandro Etalle, Pieter Hartel, and Emmanuele Zambon. Poseidon: a 2-tier anomaly-based network intrusion detection system. In *IWIA '06: Proceedings of the Fourth IEEE International Workshop on Information Assurance*, pages 144–156, Washington, DC, USA, 2006. IEEE Computer Society.

[7] Damiano Bolzoni, Sandro Etalle, and Pieter H. Hartel. Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In *RAID '09: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, pages 1–20, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] Tobias Chyssler, Stefan Burschka, Michael Semling, Tomas Lingvall, and Kalle Burbeck. Alarm reduction and correlation in intrusion detection sys-

tems. In *Detection of Intrusions and Malware & Vulnerability Assessment, GI SIG SIDAR Workshop*, pages 9–24, 2004.

[9] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 202 – 215, 2002.

[10] Oliver Dain and Robert K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *In Proceedings of the 2001 ACM workshop on Data Mining for Security Applications*, pages 1–13, 2001.

[11] Dick de Ridder Ferdinant van der Heijden, Robert Duin and David M. J. Tax. *Classification, Parameter Estimation and State Estimation: An Engineering Approach Using MATLAB*. WileyBlackwell, September 2004.

[12] Abdoul Karim Ganame, Julien Bourgeois, Renaud Bidou, and Francois Spies. A global security architecture for intrusion detection on computer networks. *Computers & Security*, 27(1-2):30 – 47, 2008.

[13] Ashish Gehani and Gershon Kedem. Rheostat: Real-time risk management. In *Recent Advances in Intrusion Detection*, pages 296–314. Springer, 2004.

[14] K. Chidananda Gowda and E. Diday. Symbolic clustering using a new dissimilarity measure. *Pattern Recogn.*, 24(6):567–578, 1991.

[15] Kjetil Haslum and A. Å rnes. Multisensor real-time risk assessment using continuous-time hidden Markov models. *Computational Intelligence and Security*, pages 694–703, November 2006.

[16] Wei He, Chunhe Xia, Haiquan Wang, Cheng Zhang, and Yi Ji. Game Theoretical Attack-Defense Model Oriented to Network Security. *Computer Science and Software Engineering,*, 2008.

[17] Kenneth L. Ingham and Hajime Inoue. Comparing anomaly detection techniques for http. In *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007*, pages 42–62, 2007.

[18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, September 1999.

[19] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[20] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, 6:443–471, 2003.

[21] Christopher Kruegel, Giovanni Vigna, and William Robertson. A multi-model approach to the detection of web-based attacks. *Comput. Netw.*, 48(5):717–738, 2005.

[22] United States et al. V. Carroll Towing Co., Inc., et al., 1947.

[23] Weiming Li and Zhengbiao Guo. Hidden Markov Model Based Real Time Network Security Quantification Method. *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, pages 94–100, 2009.

[24] C. Manikopoulos and S. Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *Communications Magazine, IEEE*, 40(10):76 – 82, October 2002.

[25] Peter Mell, Karen Scarfone, and Sasha Romanosky. A complete guide to the common vulnerability scoring system version 2.0. *Published by FIRST-Forum of*, pages 1–23, 2007.

[26] Kent D Miller. A framework for integrated risk management in international business. *Journal of international business studies*, 23(2):311–331, 1992.

[27] Peng Ning, Yun Cui, and Douglas S. Reeves. Analyzing intensive intrusion alerts via correlation. In *In proceedings of the 5th International symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 74–94, 2002.

[28] Peng Ning and Dingbang Xu. Learning attack strategies from intrusion alerts. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 200–209, New York, NY, USA, 2003. ACM.

[29] Peng Ning and Dingbang Xu. Hypothesizing and reasoning about attacks missed by intrusion detection systems. *ACM Transactions on Information and System Security*, 7:591–627, 2004.

[30] National Bureau of Standards. Guideline for automatic data processing risk analysis, 1979. Federal information processing standards publication; 65.

[31] Kevin John Soo Hoo. *How much is enough: a risk management approach to computer security*. PhD thesis, Stanford University, Stanford, CA, USA, 2000. Adviser-May, Michael M.

[32] Anna Sperotto, Ramin Sadre, F. van Vliet, and Aiko Pras. A Labeled Data Set For Flow-based Intrusion Detection. *IP Operations and Management*, pages 39–50, 2009.

[33] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. *J. Comput. Secur.*, 10(1-2):105–136, 2002.

[34] Gary Stoneburner, Alice Goguen, and Alexis Feringa. Risk Management Guide for Information Technology Systems, 2002. `http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf`.

[35] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*, pages 31–38. ACM Press, 2000.

[36] the International Organization for Standardization. Information technology - security techniques - information security management systems - overview and vocabular, 05 2009.

[37] Alfonso Valdes, , Alfonso Valdes, and Keith Skinner. Probabilistic alert correlation. In *In Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 54–68, 2001.

[38] Frank van Vliet. Turnover Poseidon : Incremental Learning in Clustering Methods for Anomaly based Intrusion Detection. In *4th Twente Student Conference on IT*, Enschede, 2006.

[39] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

[40] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors, *Recent Advances in Intrusion Detection*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-30143-1_11.

Web References

[41] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996. http://www.ietf.org/rfc/rfc1945.txt.

[42] Bro intrusion detection system - bro overview, 07 2010. http://www.bro-ids.org.

[43] The Mitre Corporation. Cwe/sans top 25 most dangerous software errors, 9 2010. http://cwe.mitre.org/top25/.

[44] Rohit Dhamankar, Mike Dausin, Marc Eisenbarth, James King, Wolfgang Kandek, Johannes Ullrich, Ed Skoudis, and Rob Lee. Sans: The top cyber security risks, 9 2009.

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817, http://www.ietf.org/rfc/rfc2616.txt.

[46] Gordon Lyon. Top 5 intrusion detection systems, 2006. http://sectools.org/ids.html.

[47] Modsecurity: Open source web application firewall, 07 2010. http://www.modsecurity.org.

[48] M. Curtis Napier. Gentoo security handbook, 07 2005. http://www.gentoo.org/doc/en/security/.

[49] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260, http://www.ietf.org/rfc/rfc2474.txt.

[50] Welcome to the home of ossec, 07 2010. http://www.ossec.net.

[51] Alientvault - creators of ossim - the oss correlation and security suite, 08 2010. http://www.alienvault.com.

[52] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349, http://www.ietf.org/rfc/rfc791.txt.

[53] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, http://www.ietf.org/rfc/rfc793.txt.

[54] FreeBSD Documentation Project. Freebsd handbook, chapter 14 security, 2010. http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/security.html.

[55] The Open Web Application Security Project. Owasp top 10 - 2010: The ten most critical web application security risks, 4 2010. http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[56] Robert Richardson. 2008 csi computer crime & security survey, 2008. i.cmpnet.com/v2.gocsi.com/pdf/CSIsurvey2008.pdf.

[57] Martin Roesch and Brian et al. Caswall. web-attacks.rules, 2 2005. http://cvs.snort.org/viewcvs.cgi/snort/rules/web-attacks.rules.

[58] Pine Digital Security. Quickscan rapport. Technical report, Pine Digital Security, Loire 130, The Hague, Netherlands, 04 2010.

[59] Snort :: Home page, 07 2010. http://www.snort.org/.

[60] Frank van Vliet. Certified secure web hacking principles - authentication and authorization. Technical report, Certified Secure, Loire 128A, The Hague, Netherlands, jan 2010.

[61] Frank van Vliet. Certified secure advanced server scan checklist. Technical report, Certified Secure, Loire 128A, The Hague, Netherlands, January 2010. https://www.certifiedsecure.com/checklists/cs-advanced-server-scan.pdf.

[62] Frank van Vliet. Certified secure advanced web application scan checklist. Technical report, Certified Secure, Loire 128A, The Hague, Netherlands, January 2010. https://www.certifiedsecure.com/checklists/cs-advanced-web-application-scan.pdf.

[63] Frank van Vliet. Certified secure basic server scan checklist. Technical report, Certified Secure, Loire 128A, The Hague, Netherlands, January 2010. https://www.certifiedsecure.com/checklists/cs-basic-server-scan.pdf.

[64] Frank van Vliet. Certified secure basic web application scan checklist. Technical report, Certified Secure, Loire 128A, The Hague, Netherlands, January 2010. https://www.certifiedsecure.com/checklists/cs-basic-web-application-scan.pdf.

[65] R. Zuccherato and M. Nystrom. ISO/IEC 9798-3 Authentication SASL Mechanism. RFC 3163 (Experimental), August 2001. http://www.ietf.org/rfc/rfc3163.txt.