ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Matthias J. Egli

Security assessment of infection incidents in the ETH university campus

Master Thesis MA-2011-03
March 2011 to September 2011

Supervisor: Elias Raftopoulos
Co-Supervisor: Dr. Xenofontas Dimitropoulos
Professor: Bernhard Plattner

Abstract

Infections of the end user's IT systems have become a lucrative goal for cybercriminals. Many types of malware are used and constantly multiple new variants are developed with increasingly stealthier behavior. As a consequence, the task to detect the diverse types of malware is becoming progressively harder.
In this master thesis a framework which collects and correlates signs of infections was developed. Multiple sources of security relevant informations were analyzed and used to provide forensic evidence of high confidence.

The Intrusion Detection System SNORT provided the basic informations about suspicious behavior of a host. This information was enriched by querying blacklists and a web search engine about the contacted hosts in the Internet. The operating system and running services on the host were detected through network scans. All features were made available through a web-based security dashboard. An automated classification based on a decision tree enabled the quick analysis of new incidents.

During two periods of 41 and 7 consecutive days 316 suspicious hosts were assessed. 31 Trojan infected hosts, 136 Spyware infections and 124 clean hosts were found. The remaining 25 cases remained unknown. In 51% of the cases two methods and in 20% of the cases 3 methods had to be combined to guarantee high confidence in the classification of a suspicious host. The automated classification could correctly analyze 83% of all infections.
A labeled data set of security incidents based on realistic traffic from a medium sized network was created and can be used for future research.

Concluding, the combination of multiple methods to reliably infer an infection is necessary in most cases. Classification algorithms can help to classify many incidents, but their success rate is too low for a fully automated security assessment.

Acknowledgments

First I would like to thank my tutor Elias Raftopoulos for the many hours he invested in my master thesis. His input and the constant availability to discuss any interesting problem were greatly appreciated. I would also like to thank the people who gave me insightful comments and input: Dr. Xenofontas Dimitropoulos, David Gugelmann and Hubert Ritzdorf. To Brian Trammell, Dominik Schatzmann and Prof. Eduard Glatz my thanks for providing the many helpful tools and access to valuable data.

Lastly I would like to thank my parents. Enabling me to study all this time and providing help whenever it was needed was invaluable. I am honored to be able to dedicate this thesis to them.

Contents

1 Introduction **1**
- 1.1 IT Security and Threats . . . 1
- 1.2 Motivation and Problem Statement . . . 2
- 1.3 Goals . . . 3
- 1.4 Structure of Thesis . . . 4
- 1.5 Related Work . . . 4

2 State of the Art in Security Incident Evaluation **5**
- 2.1 Host based methods . . . 5
  - 2.1.1 Anti-virus / Anti-spyware . . . 5
  - 2.1.2 Personal Firewalls . . . 6
  - 2.1.3 Software Inspectors . . . 7
  - 2.1.4 Forensic Tools . . . 8
- 2.2 Network Based Methods . . . 9
  - 2.2.1 Intrusion Detection Systems (IDS) . . . 9
  - 2.2.2 Host Scanning . . . 11
  - 2.2.3 Vulnerability Scanner . . . 13
  - 2.2.4 IP Traffic Data - NetFlow . . . 15
  - 2.2.5 Network Firewall . . . 17
- 2.3 Information Gathering . . . 19
  - 2.3.1 Security Log File Analysis . . . 19
  - 2.3.2 Public Vulnerability Resources . . . 21
  - 2.3.3 Blacklists and Malware Trackers . . . 22

3 Methodology **23**
- 3.1 Environment . . . 23
- 3.2 Feature Extraction . . . 24
  - 3.2.1 Intrusion Detection System Snort . . . 24
  - 3.2.2 Vulnerability Scanner OpenVAS . . . 27
  - 3.2.3 Blacklists . . . 28
  - 3.2.4 Nmap Network Mapper . . . 31
  - 3.2.5 DNS host name . . . 33
  - 3.2.6 Google Automatic Querying . . . 34
- 3.3 Feature Correlation . . . 37
  - 3.3.1 Classification Algorithms . . . 37
  - 3.3.2 Implementation . . . 42

4 Host State Assessment **44**
- 4.1 Back end with Python and Flask . . . 44
  - 4.1.1 External Modules . . . 44
  - 4.1.2 Architecture . . . 45
- 4.2 Web front end . . . 47

5 Results **49**
5.1 Overview . . . 49
5.2 Security incidents . . . 50
5.2.1 Incident Classes . . . 51
5.2.2 Bot Infections . . . 51
5.2.3 Common False Positives . . . 52
5.2.4 Infected Clients behind Servers . . . 53
5.3 Method Evaluation . . . 54
5.3.1 IDS SNORT . . . 55
5.3.2 Search Engine Google . . . 55
5.3.3 Network Scanner Nmap . . . 55
5.3.4 Blacklists . . . 56
5.4 Feature correlation and classification . . . 57
5.4.1 Extracted Features . . . 57
5.4.2 C4.5 Decision Tree . . . 58
5.4.3 Comparison to other classification algorithms . . . 61

6 Discussion and Future Work **62**
6.1 Discussion . . . 62
6.1.1 Security Incidents . . . 62
6.1.2 Automated Analysis . . . 63
6.2 Future work . . . 64

7 Conclusion **65**

A Original Task **66**

B Nmap Top 50 most often used ports **69**

C OpenVAS example output **70**

D Features used for automated Classification **75**

E Keywords used to generate Tags **78**

Bibliography **79**

List of Figures

2.1 Avira Antivir Virus found Warning . . . 5
2.2 Windows Defender - anti-spyware Warning . . . 6
2.3 Windows Firewall - Request for Internet Access of new program . . . 6
2.4 Vulnerable Software on an End-Host - Secunia PSI . . . 7
2.5 Encase Forensics searching a hard-drive for textfiles . . . 8
2.6 Intrusion Detection System SNORT - Logo . . . 9
2.7 Network Scanner nmap - Logo . . . 11
2.8 Vulnerability Scanner OpenVAS - Logo . . . 13
2.9 Example Output of OpenVAS v4.0 - Part of the detailed HTML output of a host with a severe vulnerabilities . . . 14
2.10 Capturing of IP traffic data using NetFlow . . . 15
2.11 A network based firewall controlling the traffic between the Internet and the local network . . . 17
2.12 Syslog-NG - a popular open-source log daemon . . . 19
2.13 Google and ThreatExpert Logos . . . 21
2.14 Shadowserver Logo . . . 22

3.1 Overview of Snort infrastructure: Sensor, Storage and Analysis . . . 24
3.2 Process of extracting significant features from Snort logs . . . 26
3.3 Overview of the vulnerability scanning process with OpenVAS . . . 27
3.4 Used blacklists: apews.org, dshield.org, emergingthreats.org, shadowserver.org and urlblacklist.com . . . 28
3.5 Process of checking for hits in blacklists . . . 30
3.6 Example of a Nmap scan and the extracted features . . . 32
3.7 Information retrieval about malicious activity using Google Search Engine . . . 34
3.8 Naive Bayesian network extracted from security incidents from June 2011 - reduced to 5 features . . . 38
3.9 Tree augmented naive Bayesian network extracted from security incidents from June 2011 - reduced to 5 features . . . 39
3.10 *C4.5* decision tree extracted from security incidents from June 2011 - reduced to 5 features . . . 41
3.11 Process of classification using Weka and the J48 classification algorithm . . . 43

4.1 Debugger view of the embedded web server in Flask . . . 45
4.2 Overview of the architecture employed in the back end . . . 46
4.3 Screen shot of the web front ends main page, showing the form to create a new case and the decision tree . . . 47
4.4 Screen shot of the web front ends feature extraction page. All features for the investigated IP are already extracted. . . . 48
4.5 Screen shot of the web front ends search engine interface. . . . 48

5.1 Combined methods and the percentage of suspicious cases categorized by the combination . . . 54
5.2 Decision tree generated from 114 classified incidents from one week in June 2011 - confidence level 0.076 . . . 59

List of Tables

3.1 Example of collected features and the corresponding classification 37

5.1 Types of identified security incidents in period April-May 2011 and in one week in June 2011 51
5.2 Number of times a security feature extraction method was necessary to infer if an infection occurred - taken from 74 incidents analyzed in April and May 2011 . . . 54
5.3 Types of identified security incidents in period June 2011 57
5.4 Confusion matrix of classified hosts - Based on the J48 implementation of the C4.5 algorithm - Training and testing on the same dataset 60
5.5 True and False Positive Rate for the C4.5 decision tree 60
5.6 False Positive Rates for several classification algorithms based on 10-fold stratified cross-correlation - security incidents from June 2011 61
5.7 True Positive Rates for several classification algorithms based on 10-fold stratified cross-correlation - security incidents from June 2011 61

Listings

2.1 Example output as given by the SNORT IDS 10
2.2 Output of a nmap scan of a single host 12
2.3 Example of several flows, processed by Silk. The IP-Adresses have been randomized 16
2.4 Example of an IP blocked by several perimeter firewalls in a campus network - taken from ETH Komcenter List of blocked IPs 18
2.5 Failed SSH login attempt as reported in the auth.log file 20
3.1 Features found by the feature analyzer for a suspicious server 25
3.2 Blacklist hits for outgoing connections of a host 29

Chapter 1

Introduction

1.1 IT Security and Threats

Security in the domain of Information Technology (IT) is quickly moving from the realms of computer specialists (often called hackers) into the life of everyone. IT related threats become ubiquitous. Governments or government-sponsored groups attack companies [1]. Spam is sent to almost every e-mail address. Lately political activists, helped by skilled IT security specialists, take down websites [2], change their content [3] or publish classified information [4]. End-users are becoming a target, e.g. because of published password-databases [5] and password reuse. An end-user using the same password for the Playstation (a gaming device) and PayPal (a popular online payment service) is a likely target.

Protecting against these security threats becomes a highly challenging task. Not long ago the usage of an antivirus program was considered to be adequate. After the spread of worms, infecting millions of PCs in days, firewalls were advocated as a good line of defense. Nowadays the browser and the software used to view content downloaded from the web is the main point of attack. The popular “golden rule” is to keep the system and all programs updated all the time. While the former steps involved installing additional software only once, the new methodology forces the end-user to be always on guard. The past has shown that end-users are already overwhelmed by the complexity of computer programs, and excessive warnings about update requests for all kind of programs every few days will only add to the confusion.

When end-users are not capable of protecting their systems all the time against fast-moving and changing threats, the importance of a system detecting infected clients raises. These systems are called Intrusion Detection Systems (IDS). Such a system is comparable to an automatic fire alarm.
The challenge in these systems is the high degree of mutability and stealthiness of modern malware. Just as the immune system of humans is constantly learning and adjusting, these systems must be trained and adjusted to know what they have to look for. Unlike the human immune system, these systems are still not specific enough and as a consequence often configured in such a way as to raise an alarm for everything looking even slightly suspicious.

The problem of infected computers is not someone else’s problem any more. Infected computers are used to commit computer crimes without the knowledge of the owner. In the

best case, the user will be notified by his Internet provider. In a bad case, he is notified by his bank because his bank account is empty, or in the worst case by the police suspecting him to be part of an illegal operation.

1.2 Motivation and Problem Statement

Our primary motivation for this work is the necessity to detect security incidents in an IT infrastructure. For a network administrator, it is important to keep his[1] network operational. Having users in the network with infected PCs, he risks network problems e.g. caused by quickly spreading or scanning malware. As a consequence other networks may start to block traffic from the network, thereby interrupting access for all users of the infected network.
Reliably detecting malware is very hard. If every suspicious activity in a network is inspected, most cases will be false alarms. Malware is getting very good at disguising its activities. To be effective, malware detectors have to be sensitive to general suspicious behavior as it becomes very hard to track the abundance of different types of malware. This results in a very high number of alarms without an infection of the suspicious host.

Computer systems and network components usually collect an abundance of information which can theoretically be exploited to detect security problems and ultimately improve the IT security. However security sensors are normally configured to log a warning even if the underlying reason is only slightly suspicious as not to miss any potential malicious activity.
For the ETH campus, this results in more than 3 million alerts on a normal day from the Intrusion Detection System (IDS) alone. On a busy day, the number can be more than ten times higher, reaching 38 million alerts. In addition, there are alerts from firewalls, web servers, proxy-servers, DNS servers, mail servers $\cdots$. Far less than 1% of all “security incidents” are significant.

Filtering and cross-correlating this information is a way to handle the huge amount of data. By combining different methods, false positives can be filtered. The focus of the administrators can be put on the most severe cases where multiple sensors provide signs of an infection.
An example of such a situation is a host which triggers suspicious traffic because of a spelling error. Web browsers identify themselves in the User-Agent string. An IDS watches for wrongly spelling errors in this string and finds a host contacting multiple other hosts in the Internet pretending to be the *Mozila* browser instead of the correctly spelled *Mozilla* browser. We then check a number of blacklists to find if suspicious contacts to known malicious hosts exist. The blacklist finds one host which belongs to the Russian Business Network (RBN), a known network often involved in sending unwanted E-Mail advertising and other malicious activity. We use a search engine to check if we can find more information about the suspicious external host. No hits referring to malware are found, but the search engine is reporting connections to Peer-to-Peer File Sharing (P2P) networks. A check for the hostname of the suspicious external host shows that it belongs to the popular file sharing platform *thepiratebay.com*. We conclude that the suspicious host in our network is using an anonymization tool which changes the User-Agent string to access this page. The host is likely not infected, as there is another plausible explanation for the generated alarms.

[1] The male expression is used only for readability reasons, both female and male persons are implicitly addressed

This example explains our motivation to combine multiple approaches to filter out as many false positives as possible. This enables the IT department to focus on the serious cases of infections.

An efficient handling of the security incidents is needed to help the network and security administrators in their daily work. Repetitive work has to be avoided since it demotivates and ultimately de-sensibilizes the employee. Instead the knowledge of the underlying techniques and methods of security incidents has to be built up. Only by understanding the reasons why an an automated system has come to a specific conclusion can a human adjust and refine the system. New threats with new attack patterns are emerging daily, and only a flexible system can cope with the demands.

The security research community is hindered by the lack of labeled data sets of security incidents. Such a data set should ideally contain very realistic traffic and at the same time a list of infections in the network and the timespan in which these infections were active.
Generating this data is very time consuming. It is still worth the effort, as this labeled data set can be used to measure the performance of many new malware detectors. We try to provide such a data set in the hope that it will be beneficial for future research.

1.3 Goals

The first goal of this work is to identify, classify and evaluate the sources of information about computer security. By manually evaluating the usefulness of the sources described in the problem statement, promising methods should be identified.

In the next step, these methods should be automated. An infrastructure will be built in which a computer system can automatically collect the required information. The system must collect and store information about ongoing and past infections to provide highly significant forensic evidence. A software middleware must be provided which will power the user interface frontend used by a network administrator. This middleware should be a modular and flexible to allow to easily extend the functionality.

A visualization tool that will operate as a dashboard should be developed. It will provide the security operator with all the information required in order to investigate suspicious nodes. The data processed by the middleware is displayed in an intuitive way, enabling the operator to further dive into the details of a specific problem.

The final goal is to assess suspicious hosts found in the network of the university. The state of these hosts should be identified to classify them either as being infected or as being clean. Additional information like the time span of the infection and reasons for the drawn conclusion further increase the usability of this data set. A practical application which should be done during the thesis is the assessment of a new method to detect security incidents based on SNORT data.

1.4 Structure of Thesis

The thesis is structured in the following way: In Chapter 2 the state of the art in detecting security incidents is presented, with a focus on network based detection methods. In Chapter 3 the methodology is described in 3 parts: the first part describes new and existing tools to extract information (features) about the behavior of the suspicious host. In the second part the features are correlated with an interpretable classification method. In the third part the type of infection is inferred based on the extracted features. In Chapter 4 we describe the implementation details of the developed security assessment platform. The results of the evaluation of the data collected during a 5 week period are presented in Chapter 5. A discussion of the results and promising future work is described in Chapter 6. Finally the conclusion and the main findings are presented in Chapter 7.

1.5 Related Work

A rich variance of literature exists for IT security. The following books give an overview of methods useful to find infections in an IT environment: “End-to-End Network Security: Defense-in-Depth” by Omar Santos [6] focuses on proactive security, but also handles network forensics. “Network Security Assessment: Know Your Network” by Chris McNab [7] provides a highly useful overview of tools, weaknesses and strengths of the tools and has the goal to provide a systematic way to network security assessments.

Providing a unified interface to collect and correlate security informations from various sources is the goal of the *Security Event Manager* (SIEM) Alientvaults [8]. An open source version (OSSIEM [9]) exists, which has all features of the commercial edition except for a storage system for the forensic data. It is used in multiple networks. OSSIEM helps to visualize all the activities going on in your network, but the correlation engine needs to be manually configured by the network administrator. Our approach focuses on automated correlation and can be used to extract rules for OSSIEM correlation.

Using SNORT [10] to find security incidents is the topic of many publications. A master thesis focusing on Extrusion Detection with SNORT was done by Cécile Lüssi in 2008: “Signature-based Extrusion Detection”[11]. A behavior-based approach is suggested to filter the high number of false positives. In this thesis SNORT is used as a source of extrusion data (outbound contacted IPs and ports) for further analysis. The prior task of finding suspicious hosts in this huge set of data is not part of this thesis. The primary used method of extraction is described in Elias Raftopoulos work on “Detecting, Validating and Characterizing Computer Infections from IDS Alerts” [12]. A part of this master thesis was dedicated to validate the results achieved by this detection method.

The generation of a labeled data set to test intrusion detection systems was, to the best of my knowledge, only done once before: in 1999 and 2000 by DARPA [13]. In this work background traffic and several attacks were simulated and recorded. The recorded traffic could then be used as input for tests and evaluations of IDS systems. However the artificial generation of the traffic and the nowadays unused attack patterns received a lot of criticism [14].

Chapter 2

State of the Art in Security Incident Evaluation

Multiple tools are available to reconstruct the way an intruder managed to break into the network and to infect computers (Forensics). In the following the important tools will be described. We will focus on the network based intrusion detection methods, since a direct access to the infected hosts is very often not possible. The major goal of the subsequent description and the corresponding examples is to clarify whether a system was infected. To accomplish this, we focus on outbound traffic, originating from potentially infected hosts inside the network.

2.1 Host based methods

For all Host-based methods either a pre-installed program or physical access to the PC is needed. Although this was not possible in the current analysis, the host-based methods are listed nevertheless in order to give a complete overview.

2.1.1 Anti-virus / Anti-spyware

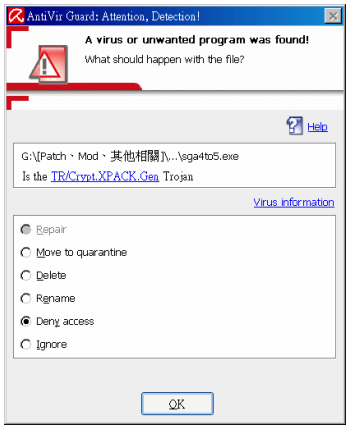


Figure 2.1: Avira Antivir Virus found Warning

The most common type of host-based protection is an anti-virus program. This kind of protection is almost exclusively deployed on MS Windows based operating systems. The antivirus program constantly scans for suspicious byte signatures in the files by comparing them to a signature database. These signature databases are normally updated daily to incorporate new viruses. If a program or file is found on the computer which triggers one of the signatures, access to this program is blocked and the user is warned. The signatures are highly specific, resulting in almost no false positives (i.e there are no warnings of a virus when there is no actual infection). The reason is that triggering a warning on benign software can have serious consequences [15]. The downside is that it is relatively easy to evade the anti-virus

program: by changing the *malware* only slightly, it wont be detected any more. Nowadays modern malware can change its appearance dynamically, therefore making a detection very difficult. Consequently the detection rate of virus scanners tend to drop recently.

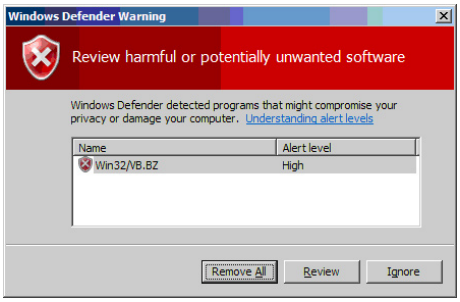


Figure 2.2: Windows Defender - anti-spyware Warning

A virus is a software which was installed in an obviously malicious way, used to commit computer crimes like attacking websites or sending spam E-Mails. There is another class of *malware* , called *spyware* . The user has installed it accidentally without knowing that the software will report the users activity to a company, which will make profit by offering customer-tailored advertising. Such *spyware* misses features of a virus like the self-replication functionality. Nevertheless, it is considered malicious by many users.
To find and remove this kind of software, anti-spyware is deployed. An example of the most commonly used anti-spyware can be seen on the left. This kind of software is also highly specific, but the challenge is drawing the line between benign software (e.g. if a user wants to be tracked to receive other benefits) and unwanted software. From an IT security perspective, this software is not problematic, but from the point of view of a company, which does not want its employees to be watched by an external party, these programs are clearly malicious.

Benefit

It would be very helpful to have a central location, where all anti-virus or anti-spyware reports from the different computers are collected. Because of its low false positive rate, a virus found by an anti-virus engine is a very good indication of a security incident. The lack of such an infrastructure for the fixed PCs and the impossibility of creating such an infrastructure for privately owned mobile student/employee laptops means that we cannot use this option. And even if such an infrastructure would exist, it would not catch new malware which spreads actively.

2.1.2 Personal Firewalls

Figure 2.3: Windows Firewall - Request for Internet Access of new program

Firewalls control the access from and to the network. They can be deployed at different levels of the network infrastructure, e.g. on the end-host or in the routers connecting countries like in the case of the “Great Chinese Firewall”. Here we focus on the end-host based firewalls.
Firewalls are based on rules defining which program or port may be accessed from which part of the network. Programs which are only needed in the local network, like file sharing or printing services, can be blocked for external hosts so that the access from the Internet is not possible. Firewall rules are highly diverse, ranging from a “Drop all” (nobody may connect from the internet) to a “Allow all” policy.

A warning is logged whenever programs violate a firewall rule (e.g. someone trying to connect to the printer service from the Internet). In addition, good firewalls can detect common malicious activity like end-host scanning , which are used to find services which can be attacked. Another interesting feature of firewalls is their ability to find programs on the end-host which are trying to connect to other PCs even if they shouldn't do so.
Firewalls come with their own set of problems: they have to be configured manually and a good configuration requires the knowledge of all programs which need to communicate with the internet. Often the "solution" of a problematic internet connection is to disable the firewall. P2P programs like BitTorrent or Skype also engage in scanning to find other users. They can cause false positive signals.

Benefit

Personal firewalls on an end-user system are very useful to fight against many worms. It can be used to detect break-out attempts by newly installed malware, at least as long as the malware does not affect the locally installed firewall (i.e if the malware couldn't gain administrative rights). In general personal firewalls are of little use for assessing if a successful infection was done, as malware can circumvent these firewalls. They need to be manually configured on a per-host basis, which is error-prone. A more promising method is to use network based firewalls which are immune to local malware. They are also capable to detect scans which are not limited to one host.

2.1.3 Software Inspectors

Figure 2.4: Vulnerable Software on an End-Host - Secunia PSI

Nowadys (2011) end-users must ensure to always have updated software installed on their PC. Without (and sometimes even with) the latest security patches, surfing in the internet can be dangerous. Drive-by downloads are becoming the dominant way of malware to infect PCs. Drive-by downloads is a term referring to automatically download and sometimes install programs when surfing the web. Often social engineering is used to trick the user into installing a certain program.
Fortunately so called software inspectors exist which automatically check the installed versions of programs on a PC and warn if a dangerous security hole is known to exist in a program.
This software is often used in managed environments such that system administrators can keep track over all installed programs and centrally start an update without the need to walk by each client PC. The logs can be used to cross-correlate against other malware signatures. If a malware is known to propagate using a certain program (e.g. Adobe Flash), but the program was never installed on the PC, an alarm from this malware is likely a false positive.

Benefit

Logfiles from *software inspectors* can help in evaluating the security state of a system and to reduce the false positive rate by cross-correlation. They are mainly used in incident prevention. After having identified a successful attack against a certain software these logfiles can help in quickly identifying other vulnerable systems. However beyond their cross-correlation capabilities their potential in checking if a PC is infected is limited.

2.1.4 Forensic Tools

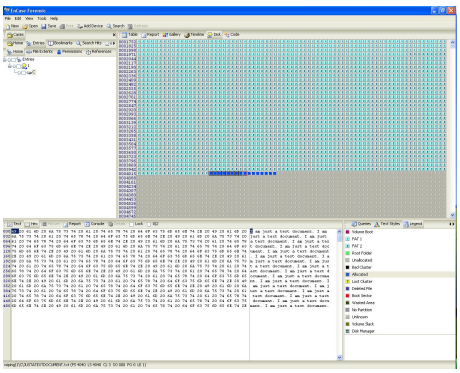


Figure 2.5: Encase Forensics searching a hard-drive for textfiles

After a computer was infected and the infection was detected, it is time for the *forensic analysis*. These tools investigate the hard-drive and, if possible, also the volatile RAM memory usually after having saved a bit-wise copy of data from the infected system to external media. They also try to identify processes which hide from the normal process manager and they look for suspicious in these processes.
A thorough forensic investigation is very difficult and expensive. It is only done for serious incidents.
Forensic analysis in an automated fashion is done by programs like ThreatExpert [16]. By taking a snapshot of the system before infection and comparing it to the situation after the program was started, the behavior can be analyzed.

Benefit

Forensic analysis is a difficult and manual task. While it can give highly interesting inside into the way a malware operates, the benefit is greatly out-weighted by the cost. Automatic forensic analysis to investigate suspicious programs on the other hand is efficient and used actively.

2.2 Network Based Methods

Network based methods to find security incidents are based on the analysis of the network traffic from the end-host. The advantage of this approach is that no individual software on the end system is needed in order to collect the data. In addition, by watching a much bigger population, anomalies in the communication between the end-hosts can be found.
However network based methods also have their disadvantages. Encrypted network traffic is problematic for network based systems since the analysis of the content (payload) of the traffic is not possible. Malware which is spreading slowly and carefully can easily hide in the background traffic. On the other side benign software can show malicious signs, e.g. by searching too aggressively for other hosts using the same software, or by causing a high load on the network.
In the following the different methods of network based methods to find infected hosts are discussed. An example of the output of the method is given, and the advantages and disadvantages are discussed.

2.2.1 Intrusion Detection Systems (IDS)

Figure 2.6: Intrusion Detection System SNORT - Logo

Network based intrusion detection is based on *event* monitoring in computer networks. The events are generated by analyzing the network traffic for signatures of possible incidents. These signatures can have different causes: they describe security critical patterns in network traffic, unacceptable or unwanted usage patterns or simply bad security practices.
An example of a security critical pattern is the request to download a software from a website already known to distribute malware. Unacceptable or unwanted traffic can be peer-to-peer (P2P) traffic which is often used for illegal file sharing. Another case of unwanted traffic can be the usage of services like Skype. Examples of bad security practices are: a request to a page with an old or insecure encryption method or the transfer of passwords in unencrypted channels.
The task of an IDS is to [17]:

1. Identify possible security incidents
2. Log the security incident in a suitable form
3. Report the incident to a security administrator

A special version of an IDS is an IPS: Intrusion Prevention System. In addition to logging suspicious traffic passively, it actively changes network traffic or configurations of other network devices like firewalls to make attacks inefficient.
IDS are widely used and a part of many network security strategies [17]. The most often used system is the open source program SNORT [10]. Two rulesets are deployes at the SNORT sensor in our network: The Sourcefire[18] ruleset from the developing company and the EmergingThreats[19] ruleset from the open source community project. For both rulesets commercial and free editions are available. The commercial rulesets usually offer more recent signature patterns and are stronger focused on malware.

```
[**] [1:2012686:1] ET TROJAN SpyEye Checkin version 1.3.25 or later [**]
[Classification: A Network Trojan was detected] [Priority: 1]
06/26-23:09:36.012392 129.132.X.Y:1052 -> 91.211.A.B:80
TCP TTL:62 TOS:0x0 ID:60960 IpLen:20 DgmLen:646 DF
***AP*** Seq: 0x3E928BAE Ack: 0x32A8617A Win: 0xFF3C TcpLen: 20


[**] [1:2010144:5] ET P2P Vuze BT UDP Connection (5) [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1]
06/27-00:09:58.405053 129.132.X.Y:11797 -> 62.149.A.B:3395
UDP TTL:126 TOS:0x0 ID:13318 IpLen:20 DgmLen:44
Len: 16


[**] [119:19:1] (http_inspect) LONG HEADER [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
06/27-08:01:19.725972 129.132.X.Y:1486 -> 83.140.A.B:80
TCP TTL:124 TOS:0x0 ID:52142 IpLen:20 DgmLen:1294 DF
***AP*** Seq: 0xDBE35365 Ack: 0x804848CB Win: 0xFB0F TcpLen: 20
```

Listing 2.1: Example output as given by the SNORT IDS

More popular experimental rulesets like the Bleedingsnort rules [20] exist. Multiple security companies use adjusted SNORT rules in their infrastructure based on security incidents which they discovered internally.
The main problem of today's IDS systems is the high number of false positives. An alarm raised by an IDS is in 99% of all cases a false alarm. The reason is not in unspecific or wrong rules, but in rules for all kind of anomalies.
Therefore it is necessary to append a compute intensive post-processing phase to the first alarm generating phase. The alarms are filtered, correlated and subjected to anomaly detection. Often an IDS is used in a system with a specific purpose, such that all traffic not fitting into this pattern (e.g. a webserver should only serve webpages and normally does not chat with other internet users) can be detected. In those cases the high number of rules is an advantage.

Example Output

In 2.1 an example of 3 logs is given. The first log is triggered by a rule describing the behavior of a Trojan (a self-replicating computer program trying to steal user information and attack other systems). The second incident origins from a rule that describes a policy Violation: The host used a well-known program to access the Bit Torrent P2P network. The last log refers to one of many pre-processing warnings: If the content of a network transmission does not follow certain rules (in this case it is too large), SNORT will stop processing the data packet and throw a warning.

Benefit

Intrusion Detection Systems are an important part of many network security strategies. Often an IDS is the first system to raise an alarm. Afterwards, several other systems are used to find additional information about the incident to reduce the high number of false positives. While an IDS alone is useless because of the extreme number of false positives, it can be very beneficial after an incident was detected for forensic purposes. Together with an IPS it can make life much harder for intruders. By writing new rules, a security administrator can incorporate attack patterns which were newly detected.

2.2.2 Host Scanning

Figure 2.7: Network Scanner nmap - Logo

A *port scanner* is used to identify services and open ports on hosts in the network. The application, such as a webserver (e.g. Apache) or a Voice-over-IP application like Skype can be identified in this way. It is good practice to run these scans in the network of an organization to get an idea on what is going on in the network and to find potentially vulnerable services. The output of the scans can be used for penetration testing, where one tries to exploit known security holes in the services.

Scanners operate by opening a connection to the port of a specific server. Because of the three-way-handshake of a TCP-connection a service listening on a specific port should answer such a request. If no service is active on the scanned port, the TCP protocol defines that a packet should be returned indicating that there is no service listening on that port. This behavior is often changed by firewalls in order to make scanning more difficult.

In addition to scanning for available services on a host, the information can be used to identify the operating system. Certain ports like TCP 135,139 or 445 are typical for a host running MS Windows (or Linux with the Samba service enabled). Other indicators are the TCP packet sequence “random” number generation or delays between packets. Combining these signatures is called *OS fingerprinting* and used by many scanners. Of course this OS scanning can be tricked by systems answering as if they were another system. By using multiple signatures, the detection success rate can be increased. OS fingerprinting on the service level is called *Service Identification*. The easiest, but also most unreliable, way to identify a service is by looking up the list of common ports as assigned by the IANA [21]. If for example an open TCP port 80 is found, the scanner will report that a webserver was found. Of course this can be easily forged, e.g. by Skype which tries to use port 80 for incoming connections as many firewalls do not block access to this port. To increase the reliability of the service identification, a scanner compares the response of the service to a database of known responses. In addition it can use a strategy called “Banner Grabbing” in which the scanned application reports its name and sometimes even the version of the software back to the scanner.

One of the most popular tool to scan a network or a host is the open-source program *nmap* [22]. It supports all the mentioned scanning techniques. Additional features are firewall evasion (SYN/FIN and XMas scans) and stealth scans. An example of its output is given below. It will not give vulnerability informations, which is the next step in a security assessment, but it identifies the services which need to be scanned by a vulnerability scanner (see 2.2.3).

Example Output

Nmap is used to scan whole networks for active hosts and, once an active host was found, it is used for an in-depth scan of the host. In this case (see 2.2), a single host was scanned thoroughly using OS fingerprinting and Service Identification. The analyzed host is a Mac OS X server, hosting multiple services like a mail-server (Postfix), IMAP & POP3 servers to distribute mail to clients (Cyrus imapd & pop3sd), a fileserver (Samba) and other services used in a organization (LDAP, KeyServer). The gained knowledge is used in two ways: It helps the security

```
Starting Nmap 5.21 ( http://nmap.org ) at 2011-05-13 12:46 CEST
Nmap scan report for xxx.ethz.ch (129.132.x.y)
Host is up (0.00071s latency).
Not shown: 64956 filtered ports, 551 closed ports
PORT      STATE SERVICE            VERSION
21/tcp    open  ftp                Mac OS X Server ftpd
25/tcp    open  smtp               Postfix smtpd
80/tcp    open  http?
106/tcp   open  pop3pw             ApplePasswordServer pop3 password change daemon 10.4.5.0
110/tcp   open  pop3               Cyrus pop3d 2.2.12 (Mac OS X 10.4.8)
139/tcp   open  netbios-ssn        Samba smbd 3.X (workgroup: GEOBOT)
143/tcp   open  imap               Cyrus imapd 2.2.12 (Mac OS X 10.4.8)
311/tcp   open  ssl/http           Apple Server Monitor http interface
389/tcp   open  ldap               OpenLDAP 2.2.X
407/tcp   open  timbuktu?
445/tcp   open  netbios-ssn        Samba smbd 3.X (workgroup: GEOBOT)
548/tcp   open  afp                Apple AFP (name: Geoserver; protocol 3.2; Mac OS X 10.3 - 10.5)
591/tcp   open  http               WebCompanion httpd 6.0v3 (FileMakerPro 6.0v4)
625/tcp   open  apple-xsrvr-admin?
636/tcp   open  ssl/ldapssl?
749/tcp   open  rpcbind
993/tcp   open  ssl/imap           Cyrus imapd
995/tcp   open  ssl/pop3           Cyrus pop3sd
3168/tcp  open  unknown
3283/tcp  open  netassistant?
3659/tcp  open  pop3pw             ApplePasswordServer pop3 password change daemon 10.4.5.0
5003/tcp  open  filemaker?
5900/tcp  open  vnc                Apple remote desktop vnc
8000/tcp  open  http               Apache httpd 1.3.41 ((Darwin) mod_jk/1.2.6 DAV/1.0.3 mod_ssl/2.8.31 OpenSSL/0.9.7l PHP/4.4.9)
8100/tcp  open  http               Apache httpd 1.3.41 ((Darwin) mod_jk/1.2.6 DAV/1.0.3 mod_ssl/2.8.31 OpenSSL/0.9.7l PHP/4.4.9)
9494/tcp  open  ssl/unknown
19283/tcp open  sassafras          Sassafras Key Server
Device type: WAP
Running (JUST GUESSING) : Linksys embedded (89%)
Aggressive OS guesses: Linksys BEFW11S4 WAP (89%)
No exact OS matches for host (test conditions non-ideal).
Service Info: Host:  xxx.ethz.ch; OS: Mac OS X

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 245.52 seconds
```

Listing 2.2: Output of a nmap scan of a single host

analyst to classify alerts coming from this machine as he knows its purpose. An alarm triggered by this server is often originating from a client who is using a service on this server. The second use of this information is to feed a vulnerability scanner with this information to discover vulnerable services on this server. Vulnerability scanners are often also capable of scanning a network, but nmap outperforms them in accuracy and speed.

Benefit

If a scan was successful (the host did reply at least on some open ports), it is a reliable source of information about the purpose of the host. A client will typically have almost no open ports beside those used for File-Sharing in the local network. A server has to reply on the services which it is offering, else no one can reach the service. This information will be forwarded to a vulnerability scanner whenever a vulnerable service is suspected. The main advantage in using networks scanners is the cross correlation capability. Many false positives can be filtered if the purpose of the server is known (e.g. a DNS-Server contacting blacklisted IPs often just replies to DNS-Queries coming from these IPs). On the other hand, alerts which should not be possible from such a host (like a webserver which connects to a P2P network) are more severe and need to be highlighted, even if they normally would be discarded.
A negative scan result (no open ports, perhaps even no answer) does not necessarily mean that there is no host at this IP address. Many PCs which are only used as clients are employing a personal firewall which blocks all access to the machine. In that case network scanning can not provide any insight into the host. In such a case other data sources, like collected traffic (e.g. by NetFlow) have to be used.

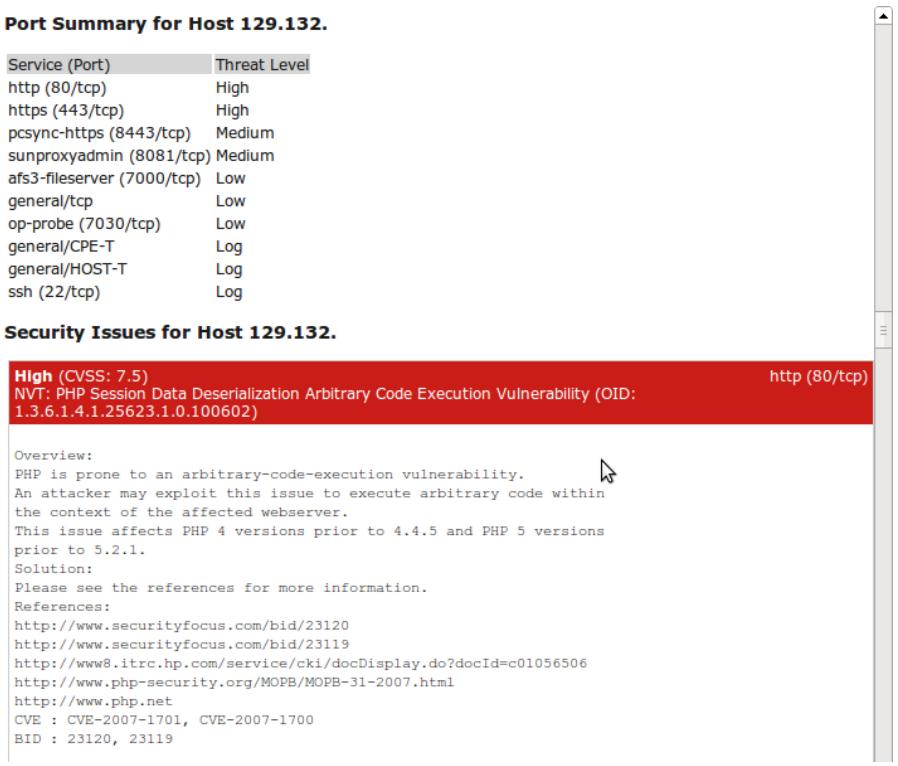
2.2.3 Vulnerability Scanner

Figure 2.8: Vulnerability Scanner OpenVAS - Logo

Vulnerability scanning can perform similar tasks as network scanning: Find active hosts and services running on these hosts. In addition they try to find vulnerabilities in these services. Often they accept the input from network scanning to speed up the process. By comparing the information received from the scan (operating System, active services) to a database of known insecure services, vulnerabilities are disclosed. Vulnerabilities can then be forwarded to "Penetration Testing Frameworks" where an actual exploit of the security hole can be performed. Some vulnerability scanners can automate this process.
Vulnerability scanners can scan a network or individual hosts. If possible, they can also scan the host by connecting to it with provided credentials, in the ideal case even with administrative rights. This scenario is useful if multiple persons have access to a host and are able to run services on it. Internal scanning from the host itself can be much quicker and more reliable. In addition policy violations can be detected both for internal and external scanning methods, e.g. someone providing a forbidden public FTP-server for file sharing.
If a vulnerability is found, ways to mitigate the problem are displayed if available. Further details about the vulnerability are referenced, such as the the date, since when the vulnerability is known, its severity and the availability of proof-of-concept code to exploit the vulnerability. The severity refers to the threat level as perceived by the designers of the vulnerability scanner. These threat levels have to be taken with care, as they can be misleading. E.g. by combining multiple low-severity vulnerabilities a skilled attacker can create a much higher threat. Or a seemingly strong threat from a public writable folder is in fact no real problem, since it is only accessible from the internal network and is used there on purpose.
The problems of vulnerability scanning is their potential disruptive behavior. They produce a lot more traffic than network scanners because they try to exploit a lot of different vulnerabilities for the same service. Some of the tests cause the service being scanned to crash. In that case a severe vulnerability is found, but also harm was done by finding the vulnerability. Sometimes the scanners cause a high number of false positives, e.g. if a patched system still reports an old version number. In general, even though these scanners provide a thorough and detailed output, they cannot assess the security state of a whole network. However they highlight interesting points of attack and may point out some serious security holes in a network.

Example Output

Vulnerability Scanners can give output in multiple forms, e.g. as executive-style reports of the overall network security, a detailed HTML document with all discovered vulnerabilities or a machine readable XML file. The example shown in Figure 2.9 is a part of the html output produced by OpenVAS v4.0, describing a severe (high risk) vulnerability which was found during a scan. The vulnerability refers to PHP, a script language used mainly to create webpages. The severity is high because an attacker can execute any code, possibly reading confidential data on the webserver or using it to attack other computers. Several other services with medium and high vulnerabilities exist.
In this case the vulnerabilities were discovered because the webserver reported the usage of an old version of PHP. Further tests showed that the webserver was not vulnerable because patches against the described vulnerability were deployed. These patches do not increase the



Port Summary for Host 129.132.

Service (Port)	Threat Level
http (80/tcp)	High
https (443/tcp)	High
pcsync-https (8443/tcp)	Medium
sunproxyadmin (8081/tcp)	Medium
afs3-fileserver (7000/tcp)	Low
general/tcp	Low
op-probe (7030/tcp)	Low
general/CPE-T	Log
general/HOST-T	Log
ssh (22/tcp)	Log

Security Issues for Host 129.132.

High (CVSS: 7.5) http (80/tcp)
NVT: PHP Session Data Deserialization Arbitrary Code Execution Vulnerability (OID: 1.3.6.1.4.1.25623.1.0.100602)

```
Overview:
PHP is prone to an arbitrary-code-execution vulnerability.
An attacker may exploit this issue to execute arbitrary code within
the context of the affected webserver.
This issue affects PHP 4 versions prior to 4.4.5 and PHP 5 versions
prior to 5.2.1.
Solution:
Please see the references for more information.
References:
http://www.securityfocus.com/bid/23120
http://www.securityfocus.com/bid/23119
http://www8.itrc.hp.com/service/cki/docDisplay.do?docId=c01056506
http://www.php-security.org/MOPB/MOPB-31-2007.html
http://www.php.net
CVE : CVE-2007-1701, CVE-2007-1700
BID : 23120, 23119
```

Figure 2.9: Example Output of OpenVAS v4.0 - Part of the detailed HTML output of a host with a severe vulnerabilities

version number, they only fix the actual problem. The vulnerability was discovered in 2007 (CVE-2007-1701), this scan was done in 2011. Old security vulnerabilities found during a scan are often patched.

Benefit

Vulnerability scanning is beneficial in giving a quick overview of the available points of attack. Without considering other problems in the network, they point to problems. As external attackers often use the same techniques, security problems discovered by vulnerability scanning should be a top priority.
For inferring if a host is infected, this technique is seldom useful. Most malware spreads by exploiting bugs in applications on end hosts which can not be detected by a vulnerability scanner. If a host shows signs of being infected by a worm[1], a scan showing if the attacked service exists on the host can be an important clue.
We did not discover any worm outbreaks during the period we evaluated and because these kind of malware is not dominant any more, vulnerability scanner are also not so important in extrusion detection.

[1] A worm is a special kind of malware which automatically spreads by attacking vulnerable services which can be accessed remotely

2.2.4 IP Traffic Data - NetFlow

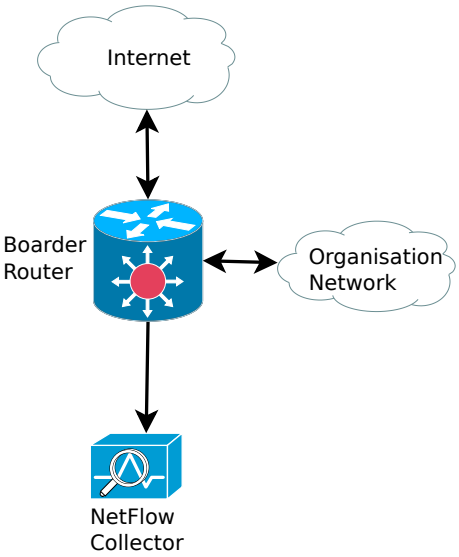


Figure 2.10: Capturing of IP traffic data using NetFlow

The Internet is based on the IP protocol (Internet Protocol). This protocol uses a packet-switched approach to deliver information. The source and destination and size of each packet can be recorded at the gateway routers which forward the packets. Often an accounting of the packet traffic is done in order to keep track of the resource usage by different customers and to bill them accordingly.
A lot of network research focuses on the analysis of this data. Cisco implemented the NetFlow v5 standard which is also widely used by other big network hardware manufacturers. NetFlow is a format in which the traffic information is exported from the routers to so called NetFlow collectors. It is the quasi-standard and NetFlow is therefore often used as a synonym for IP traffic data. The IETF (Internet Engineering Task Force) together with the main hardware vendors is developing the next standard format IPFIX [23] which will be used for IPv6 and other enhancements of the IP protocol.
Using NetFlow data the behavior of a host can be investigated without the need to scan it. However the actual payload is not captured for performance and storage capacity reasons, making deeper packet inspection as done by IDS systems impossible. The positive side effect is that encryption does not provide a problem, as the IP header is normally not encrypted. In the past a lot of anomaly detection was done with the help of NetFlow data. Worm outbreaks or network scans can be discovered by measuring the entropy in NetFlow data [24]. This allows to also discover client services which are otherwise difficult to detect by scanning (like Skype) [25]. Recent research done by Prof. Glatz demonstrates the usability of one-way flows to detect malicious activity. We will look into that topic in a later chapter.
The problem in using NetFlow data is the acquisition and handling of the data. Even for small networks, the size of NetFlow data can be substantial. When using sampled NetFlow data (every n'th packet is reported), the correct numbers must be interpolated, which can be tricky or error-prone. In addition, when using NetFlow data one has to be very careful with data protection as the privacy of the user can be violated by this data.

Example Output

The example shown in 2.3 is a list of flows with randomized IP addresses (randomized for privacy reasons). Each connection between two hosts in the network which passes the border router is recorded. The output shows that a lot of connections are using port 123 and the UDP protocol (pro = 17). This port is used by a time synchronizing service. While some connections carry a lot of packets and bytes (e.g. the first in the list), others are just consisting of one packet. This kind of traffic is often seen when a host scans other hosts.
For a serious analysis of the behavior of a host the data from this host must be filtered out of the complete traffic and subsequently the traffic has to be analyzed using several statistical methods. Algorithms exist to search for scanners, heavy hitters (users who use an unusual

sIP	dIP	sPort	dPort	pro	packets	bytes	flags	sTime	dur
114.48.192.231	59.32.109.134	123	123	17	211	16036		2011/06/20T00:00:17.207	3529.424
114.48.192.231	59.32.109.134	4002	123	17	2	152		2011/06/20T00:27:08.473	0.411
114.48.192.231	59.32.109.134	22321	123	17	2	152		2011/06/20T00:51:57.054	0.035
114.48.192.231	59.32.109.134	34039	123	17	2	152		2011/06/20T00:02:22.656	0.379
114.48.192.231	59.32.109.134	65535	123	17	227	17252		2011/06/19T23:59:56.479	3596.258
114.48.192.231	59.32.100.112	3	0	1	1	56		2011/06/20T00:02:10.174	0.000
66.48.83.219	59.32.109.134	123	123	17	9	684		2011/06/20T00:19:51.783	241.297
66.206.64.9	59.32.109.134	123	123	17	1	76		2011/06/20T00:14:47.428	0.000
66.206.64.9	59.32.109.134	123	123	17	1	76		2011/06/20T00:31:50.699	0.000
66.206.64.9	59.32.109.134	123	123	17	1	76		2011/06/20T00:48:54.545	0.000
66.159.75.149	59.32.109.134	123	123	17	1	76		2011/06/20T00:06:15.101	0.000
66.159.75.149	59.32.109.134	123	123	17	1	76		2011/06/20T00:23:19.467	0.000
66.159.75.149	59.32.109.134	123	123	17	1	76		2011/06/20T00:40:23.495	0.000
66.159.75.149	59.32.109.134	123	123	17	1	76		2011/06/20T00:57:26.958	0.000
66.182.192.41	59.32.109.134	123	123	17	56	4256		2011/06/20T00:00:47.026	3534.544
66.182.109.54	59.32.109.134	123	123	17	1	76		2011/06/20T00:09:54.534	0.000
66.182.109.54	59.32.109.134	123	123	17	1	76		2011/06/20T00:26:57.631	0.000
66.182.109.54	59.32.109.134	123	123	17	1	76		2011/06/20T00:44:02.423	0.000
66.182.11.144	59.32.109.134	40324	123	17	1	76		2011/06/20T00:12:20.735	0.000
66.182.11.144	59.32.109.134	46731	123	17	1	76		2011/06/20T00:14:20.416	0.000
66.182.11.144	59.32.228.184	64888	51413	17	1	126		2011/06/20T00:55:42.966	0.000
66.182.94.141	59.32.175.8	14932	19588	17	1	59		2011/06/20T00:36:52.822	0.000
66.182.142.142	59.32.109.134	8	0	1	273	22932		2011/06/19T23:59:41.284	3591.433
66.182.202.243	59.32.173.206	58618	80	6	6	372		2011/06/20T00:02:53.551	15.616
66.182.103.108	59.32.109.134	32769	123	17	1	76		2011/06/20T00:23:48.951	0.000
66.182.103.108	59.32.109.134	32771	123	17	1	76		2011/06/20T00:50:59.674	0.000
66.182.35.175	59.32.109.134	123	123	17	1	76		2011/06/20T00:11:56.849	0.000
66.182.35.175	59.32.109.134	123	123	17	1	76		2011/06/20T00:29:00.392	0.000
66.182.35.175	59.32.109.134	123	123	17	1	76		2011/06/20T00:46:04.379	0.000
66.182.209.54	59.32.109.134	123	123	17	1	76		2011/06/20T00:01:07.194	0.000
66.182.209.54	59.32.109.134	123	123	17	1	76		2011/06/20T00:18:11.333	0.000
66.182.209.54	59.32.109.134	123	123	17	1	76		2011/06/20T00:35:15.195	0.000

Listing 2.3: Example of several flows, processed by Silk. The IP-Adresses have been randomized

high amount of bandwidth) or denial-of-service conducting users. The implementation of these methods is challenging and time-consuming, as no ready-to-use framework exists so far. It was not done as part of this thesis.

Benefit

NetFlow data can both give interesting insight into a whole network and insight into the activities of a single host. It is possible to tell which services in the network have been used by the computer at any time, which however raises problems with data privacy laws. If the problems with data acquisition and data storage have been solved, NetFlow data is very powerful for forensic analysis. The problem is in finding the needle in the haystack. A single host can, e.g. by taking part in a P2P network, communicate with thousands of other hosts in the Internet. Finding the one malicious host in all of these IPs is almost impossible.
IP traffic data can give strong hints about malicious activity if an automatic way of extracting interesting information about a host is in place, e.g. by characterizing the amount of scanning done by this host. The downside are the very high storage and compute resources needed, which make a real-time analysis almost impossible.

2.2.5 Network Firewall

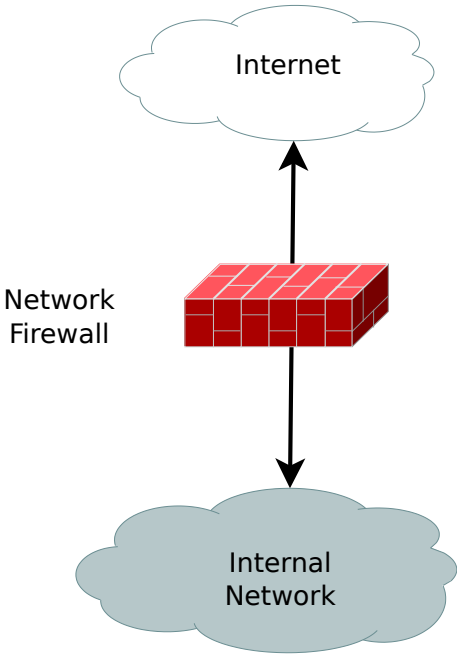


Figure 2.11: A network based firewall controlling the traffic between the Internet and the local network

Firewalls in a network are the gatekeepers and a first line of defense against intruders. A firewall controls the flow of network traffic between two hosts on opposite sides of the firewall. They are deployed at the borders of a network. This can be at the place where the network is connected to the Internet or between smaller networks inside an organization.
Firewalls help to prevent attacks using lower network layers than the application layer. Firewalls do not filter content, but only the connection itself. A typical use case is to prohibit access to any internal host except to hosts providing a public service which should be reachable from the Internet e.g. a webserver. These exceptions are normally configured manually. Often firewalls put no restrictions on outgoing traffic (traffic originating from the internal network), but in environments with high security demands (like banks) the outgoing traffic may be heavily restricted as well.
Modern firewalls can detect many low-level attacks against an infrastructure. If too many unsuccessful connection attempts from a single IP address occur, this IP address is blocked from further access to the whole network. New worm outbreaks can be detected and blocked automatically. Connections which carry a very high amount of traffic can be throttled to ensure a fair-usage policy.
While firewalls played a major role in fighting against intruders in the past (being a popular IT security topic in movies till today), the attacks have moved on from the network layer to the application layer. These kind of attacks cannot be detected or prevented by firewalls.

Example Output

The example of a log from a firewall (see 2.4) shows an external IP address which is blocked, because it connected to multiple internal IP addresses, searching for SSH servers (port 22). This alert was triggered because the attacker built up many connections to SSH servers which have been blocked at various points in the infrastructure. An automatic temporary block of this IP address was issued by correlating the results from multiple firewalls. Time-limited automatic blocks are important to reduce the number of support tickets which have to be handled by the IT support group. Often an attacker is a victim himself, being infected by a Trojan. In networks used for dialup-connections IP addresses change frequently. If an IP address continues to be blocked for a very long time, innocent users will also be affected by inheriting the IP address from an attacker.

Benefit

Firewall logs can be used to find attackers and track the connections they initiated in the network. Scan activities originating within the organization's network are also detected, giving a very strong indication that an infection or other malicious user activity took place.

```
<record id="0">
  <blocked_ip>31.3.x.y</blocked_ip>
  <blockiert_seit>2011-08-28 12:06</blockiert_seit>
  <blockiert_bis>2011-08-28 16:00</blockiert_bis>
  <block_reason>
    IP:31.3.x.y Number of Entries:170 ------------------------head ------------->Aug 28 12
    : 03 : 47 fw-mavt1-a.ethz.ch %FWSM-5-106100 : access-list acl_mdc_outside
    _access_1 denied tcp outside / 31.3.x.y (50849) -> inside / 129.132.0.99 (22)
    hit-cnt 1 (first hit) [0xf60d7cc2, 0x0]Aug 28 12 : 03 : 47 fw-mavt1-a.ethz.ch
    %FWSM-5-106100 : access-list acl_mdc_outside _access_1 denied tcp outside /
    31.3.x.y (50849) -> inside / 129.132.x.y (22) hit-cnt 1 (first hit) [0xfd62fe47,
    0x0]Aug 28 12 : 03 : 47 fw-mavt1-a.ethz.ch %FWSM-5-106100 : access-list
    acl_mdc_outside _access_1 denied tcp outside / 31.3.x.y (50849) -> inside /
    129.132.x.y (22) hit-cnt 1 (first hit) [0xf60d7cc2, 0x0]------------------tail
    ------------->Aug 28 12 : 03 : 52 asa-fw-spinoff-a.ethz.ch %ASA-4-106023 : Deny
    tcp src outside : 31.3.x.y / 50849 dst inside : 129.132.x.y / 22 by access-group
    "acl_mdc_outside _access_1" [0x8f24b0de, 0xf4a33758]Aug 28 12 : 03 : 52
    asa-fw-spinoff-a.ethz.ch %ASA-4-106023 : Deny tcp src outside : 31.3.x.y / 50849
    dst inside : 129.132.x.y / 22 by access-group "acl_mdc_outside _access_1"
    [0x8f24b0de, 0xf4a33758]Aug 28 12 : 03 : 52 asa-fw-spinoff-a.ethz.ch
    %ASA-4-106023 : Deny tcp src outside : 31.3.x.y / 50849 dst inside : 129.132.x.y
    / 22 by access-group "acl_mdc_outside _access_1" [0x8f24b0de,
    0xf4a33758]31.3.x.y 1Blocked until: 28-08-2011 16:00
  </block_reason>
</record>
```

Listing 2.4: Example of an IP blocked by several perimeter firewalls in a campus network - taken from ETH Komcenter List of blocked IPs

Firewalls are not effective against application level attacks. As most malware today uses application vulnerabilities and is distributed using drive-by-downlaods, firewalls provide only a limited protection. Infected PCs in the internal network are often not affected by firewalls as they allow internal traffic to pass unhindered. Only the most aggressive and intrusive users are found this way. Nowadays, most malware tries to be stealthy, making the detection by firewalls very hard.

2.3 Information Gathering

When the host- or network-based methods cannot identify an infection, other sources have to be used. As many different systems in a network are keeping logs about their activities, extracting relevant information for these logs can be helpful. Malware is a global problem, and most likely multiple organizations are affected by an attacker or a certain type of malware. Using public blacklists or information published by other organizations in the Internet can help to classify and evaluate an infection.

2.3.1 Security Log File Analysis

Figure 2.12: Syslog-NG - a popular open-source log daemon

A log records events triggered by an application. The logs don't have to be security-related (and most of the time they aren't). Beside the logs discussed before (from firewalls, IDS systems or antivirus engines), applications and services keep track of their activities in log files. Analyzing these log-files can give an interesting inside into activities of the service. Common types of log-files are *access logs* , keeping track of who used a service at which time, *error logs*, reporting application warnings and errors and general *activity/status logs* informing about a result of an operation.
In the following we will focus on logs which can help in finding out if a security incident has occurred. The logs from the following services are useful for this purpose:

- **DNS Server Logfiles** - disclose the actual source of a potentially malicious DNS query
- **Web Proxy Logfiles** - disclose the actual source of a request for a malicious website
- **SSH Access Logfiles** - show unsuccessful attempts to authenticate as well as the used usernames
- **Host OS Logfiles** - crashing programs (possibly attacked by malware) and the Host OS firewall can give hints at successful attacks.

Because of privacy concerns access to these logs is problematic. There is no standard log format, making an individual handling of each log a necessity. Log files are generated at multiple points in a network and must be collected and centrally stored. This not only helps to analyze them, but is also an important best practice for enabling forensic analysis after an incident was discovered. Logfiles from infected PCs have to be handled with care as the information in these logs can be forged , e.g. a trojan will normally hide logins from the attacker in the local logs.

Example Output

An example of a logfile output can only give a very small input into the world of log files. In the example shown in 2.5 a failed login attempt using the SSH (Secure Shell) protocol is displayed. It is generated on the host where the failed login attempt occured. Logs are generally saved in a textual form which can be directly parsed, but sometimes a binary format is used. Other logs are saved in a database, allowing for fast queries on the data.

```
Aug 28 17:07:34 pc-10082 sshd[22991]: Invalid user test from 84.73.x.y
Aug 28 17:07:34 pc-10082 sshd[22991]: Failed none for invalid user test from 84.73.x.y port 2531 ssh2
Aug 28 17:07:35 pc-10082 sshd[22991]: pam_unix(sshd:auth): check pass; user unknown
Aug 28 17:07:35 pc-10082 sshd[22991]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=84-73-x-y.dclient.hispeed.ch
Aug 28 17:07:35 pc-10082 sshd[22991]: pam_winbind(sshd:auth): getting password (0x00000388)
Aug 28 17:07:35 pc-10082 sshd[22991]: pam_winbind(sshd:auth): pam_get_item returned a password
Aug 28 17:07:38 pc-10082 sshd[22991]: Failed password for invalid user test from 84.73.x.y port 2531 ssh2
```

Listing 2.5: Failed SSH login attempt as reported in the auth.log file

Benefit

An infrastructure which collects, converts and stores important logfiles centrally is a very powerful tool. Beside their usage to find security incidents, they also point at other failures like hardware problems or buggy software. Logs can be used to escalate otherwise less important security warnings, e.g. if a scanner trying to gain access to a system is finally succesful. In that case an IDS will not give any more warnings (as a “correct” connection was established), but by watching for anomalies in the logs created by the machine, malicious activity can be uncovered. The main challenge is the automatic analysis of the log files. A very good filter and correlation engine must be used. Log files will generate an abundance of information, and it is very hard to filter for relevant alerts without missing critical alerts.
When a specific use case must be analyzed, the analysis of logs can be much easier than described before. For example, if for a DNS server the querying client in the internal network should be determined, this can be done in a straight-forward matter.

2.3.2 Public Vulnerability Resources

Figure 2.13: Google and ThreatExpert Logos

Vulnerability resources are informations available to (paying) administrators which help to classify a malware. Multiple forms exist: they can analyze malware automatically; they have informations about the behavior and ways of distribution of malware; they know who is distributing malware; they know which kind of malware is especially active. Today this information is mostly used to support the security administrator on his (manual) task of inferring what is going on in his network. In the following paragraph an overview of several often used freely available sources of information is presented.

ThreatExpert [16] is an automated free malware analyzer. Similar services are offered by GFI Sandbox (formerly knows as CWsandbox) [26] and Comodo Instant Malware Analysis [27]. A executable file can be transmitted, which will be executed and closely watched. ThreatExpert claims to be able to analyze malware in 2-3 minutes, which is considerably faster than a manual analysis needs. The results are archived and can be searched. Among other features (opened Files, hits by virus scanners, registry entries) the contacted IP addresses are logged. This information can help to build blacklists or classify the outgoing connections of suspicious hosts. A subcategory of these vulnerability scanners are the virus scanner services (e.g. VirusTotal). Often multiple virus engines are used to check a file. Another subcategory are the URL checkers (like http://www.ipvoid.com/) which do not scan an executable file, but a suspicious page which is trying to use browser related vulnerabilities.

Robtex [28] is a page combining multiple tools used to get information about an IP address or a domain. It also features a service which checks in multiple blacklists for the occurrence of an IP address.

CVE [29] (Common Vulnerabilities and Exposures) is a list of known public vulnerabilities in applications. Several other pages exist offering similar services. In general a description of the problem and the affected software version is published. Sometimes also a proof-of-concept code to test the vulnerability is provided.

Google is the market leader for web search engines (2011). Google offers application programming interfaces (APIs) to automatically search the web using their search engine. By searching for IP addresses of suspicious hosts or observed strings sent by malware a fast overview of the results received by other administrators in the world can be done. In addition Google by design also searches the resources mentioned before and can shorten the process of finding relevant information. The process can be automated, as first described by Ionut Trestian et al [30].

Benefit

Public malware information is an essential part of the whole analysis process. It is impossible to classify and watch suspicious files manually. Also the process often involves a manual part where the information has to be analyzed by a human, it can be automatized. We will show how this can be achieved in Chapter 3.

2.3.3 Blacklists and Malware Trackers

Figure 2.14: Shadowserver Logo

Blacklists are a list of IP addresses, networks or domains which are used to distribute malware, send spam mails or attack other hosts. Often the sources for their data are firewall logs from multiple organizations. The back-tracking of the source of spam e-mails allows to create dedicated e-mail-spam blacklists. Another source of blacklist data are so called honeypots. These are hosts which are deliberately set up in vulnerable way in order to be infected by malware. The hosts are then used to watch closely what the malware is doing in order to find the sources or command channels which it is using.
Malware trackers are projects often specialized on a single kind of malware [31] [32]. ShadowServer [33] is an example of a project where multiple botnets are tracked. They aggregate information from own information sources like honeypots with information from other sources about malware and reconstruct the network behind the malware. Modern malware is controlled by distributed *command and control* (C&C) servers. From these servers an attacker is controlling what the infected PCs should do. Malware trackers provide blacklists so that organizations can block the access to these servers. In addition, they help the people responsible for a network to find C&C servers within their network so that they can be removed.

Benefit

Botnet trackers can give a highly interesting inside into the darker side of the Internet. If communication with hosts known to host a C&C server is recorded, this is a strong sign of malicious activity. Unless a scanning service like Skype or a P2P client is responsible for this communication pattern, the host is likely infected.
Blacklists, on the other hand, are often misleading [34]. The reasons behind an entry in a blacklist are often not known and because IP addresses often stay quite long inside a blacklist, they tend to be outdated. In a huge network with a lot of traffic it is not uncommon that a lot of contacted external IP addresses are found on some blacklist without involving a security incident.

Chapter 3

Methodology

Extraction and correlation of the highly diverse signatures of an infection is a complex task. Multiple tools have to be combined and a consistent framework has to be built which is used to assess the incident. In the following sections the tools and methods used in this thesis will be presented. In addition a promising new detection method is presented which focuses on one-way flows in which the contacted host does not reply. Finally the framework used for correlation and automatic extraction of relevant security features is presented.
The focus in this thesis is on extrusion detection and the assessment of a successful infection. Extrusion detection promises to be more accurate as an attack triggering an alert is often unsuccessful and results in a false positive. Alerts originating from the internal network are statistically more significant as they indicate a successful infiltration of the internal node. For this reason we do not consider alerts from external traffic.

3.1 Environment

The setting for this thesis is a medium sized university network at the Swiss Federal Institute of Technology ETH Zurich. More than 25.000 students and employees [35] used this network in 2010. In total 2.35 Petabytes were transferred to or from the Internet in 2010, using more than 30.000 cable-based Internet connections and more than 1300 wireless access points [36]. There is no public anonymous guest access and each endpoint can be tracked back to an organization or individual. Students and employees can remotely connect to the network using VPN[1] from public access points all over Switzerland or from their homes.
To detect security incidents, the Snort IDS is deployed. The system filters all incoming and outgoing traffic to the Internet. Firewalls are deployed at the border of each subnet and track incidents individually. Each firewall is configured based on the needs of the protected subnet. Some firewalls allow all traffic to pass, others block all incoming traffic. Offending IP addresses both from inside or outside the network are automatically blocked. The NetFlow data is analyzed to find suspicious activities.
Multiple different environments exist within the network. There is a centrally managed environment, in which Unix and Windows based clients are managed. Other environments are managed by the individual users. The biggest individually managed network is the public docking network used by students with their private laptops. The central infrastructure is managed by the *Informatik Dienste* (ID), the university's IT group. Departments and institutes have their own

[1] Virtual Private Network - a technology using an encrypted channel to tunnel into the internal network

IT support groups (ISGs). The size and services offered by these ISGs varies a lot, making the overall network a very heterogeneous environment.
The huge variance of users, ranging from technically skilled security researchers to students using their laptops mainly for communication in social networks, presents a very interesting field to study security incidents.

3.2 Feature Extraction

The first step in a security assessment is to collect, filter and convert the data from the security sensors. We used 6 different sensors: The intrusion detection system *Snort* , the vulnerability scanner *OpenVAS*, blacklists from various sources, the network scanner *nmap*, the *dig* tool for DNS services and finally automatic queries to Google.

3.2.1 Intrusion Detection System Snort

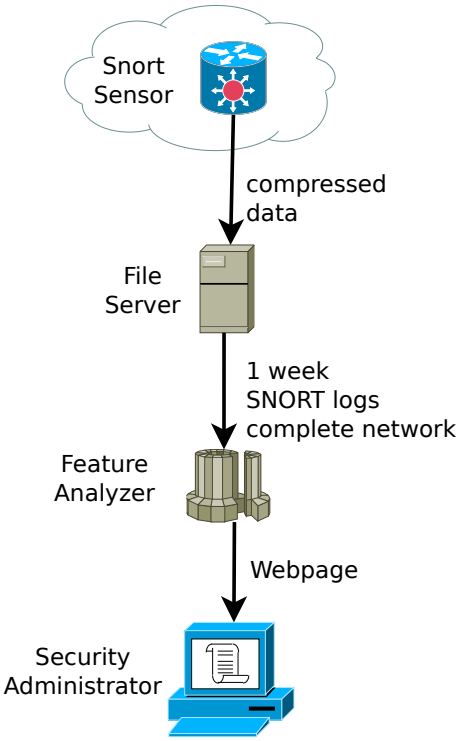


Figure 3.1: Overview of Snort infrastructure: Sensor, Storage and Analysis

Snort is the primary source of information about infected hosts. In the network Snort sensors are deployed at the gateway routers connecting the network to the Internet. They filter all incoming and outgoing traffic from the university network. The sensors are routinely used by the *Informatik Dienste* to find infections in the network. In addition the data is provided to researchers by copying it into the *Scylla Cluster*[2]
A new method to detect infections in Snort IDS data [12] allows to discover the hosts which are suspected to be vulnerable. All alerts generated by outgoing communications are evaluated in order to assess independently if these hosts are infected. In figure 3.1 the general process is visualized, while in 3.2 the detailed extraction of the features is visualized.
In the first step, a full week of compressed Snort alerts from all hosts in the network is read from the central storage server. We are only interested in alerts which are triggered by outgoing traffic from the single host which should be evaluated. To filter these alerts, the Linux tool *grep* is used, which excels at the quick analysis of ASCII encoded text files. The limiting factor for this pre-processing step is the speed of the grep tool, while the data decompression is done in parallel on another processing core with 50% of grep's CPU requirements. It is more efficient to work with compressed data, since the mean compression factor is about a factor of ten, allowing to uncompress the data faster than to read uncompressed data. On the system used for development& testing[3] the pre-processing step took 7.1 minutes, extracting data of a full week (3.1 Gb gzip compressed) from a single host. This process can be parallelized easily if the source data is split and analyzed in parallel, but since this wasn't a bottleneck in the whole analysis process no further optimization was pursued.

[2]The Scylla Cluster is a group of powerful servers used for scientific research at the ETH Zurich
[3]Intel Core 2 Quad 2.4GHz with 4GB Ram

```
dst-ip: 91.211.117.70 (26%)
dst-port: 53 (100%)
infection-time: 06-19 23:00:53 -> 06-20 14:55:13
snort: 2400001 (822, 99%) [ ET RBN Known Russian Business Network IP UDP (126) ]
src-port: HIGH (100%)
topalert-count: 822
topalerts: 2400001 [ ET RBN Known Russian Business Network IP UDP (126) ]
```

Listing 3.1: Features found by the feature analyzer for a suspicious server

After pre-processing the data analysis phase starts. The data is transferred to the analyzer using a webfrontend[4]. The data is then analyzed by a Python script which sequentially reads all lines of the log file. Snort alerts are converted into objects of a Snort python class. After conversion the following features are extracted:

- Frequently contacted hosts (>10% of all alerts reaching out to this host)
- Frequently contacted destination ports (>10% of all alerts going out to this port)
- Frequently used source ports (>10% of all alerts originating from this port)
- Count of alerts by Snort alert class (classes are “compromised”, “attack”, “scan” and “policy”, taken from [12])
- Infection duration, the time in hours between the first and the last severe Snort alert
- Occurrence of severe alerts: If an alert with high priority from the “compromised” class is found, its ID is reported as a feature
- Total count of all severe alerts

The features are stored in a *SQLite* database as a readable textual representation which can be displayed on a security administrators management console or on the webfrontend without the need of further conversion steps.

Example

In this example several connections from a server to the Russian Business Network (RBN), a network often used by computer criminals, were recorded. One single IP address is frequently related to these Snort alerts. Manual inspection confirms that this IP address (91.211.117.70) belongs to a DNS server inside the RBN. Almost all Snort alerts are referring to a communication with the RBN. The source port of all alerts is port 53, which is normally used by DNS.
After checking the hostname of the server we confirmed that this is a frequently used DNS caching server. All the DNS requests apparently originating from this host are triggered by other clients. It can be assumed that this host is not infected, but that the clients using this host are showing suspicious activity.
Listing 3.1 shows the features as they are reported by our feature detector.

[4] The details of the framework are described in chapter 4

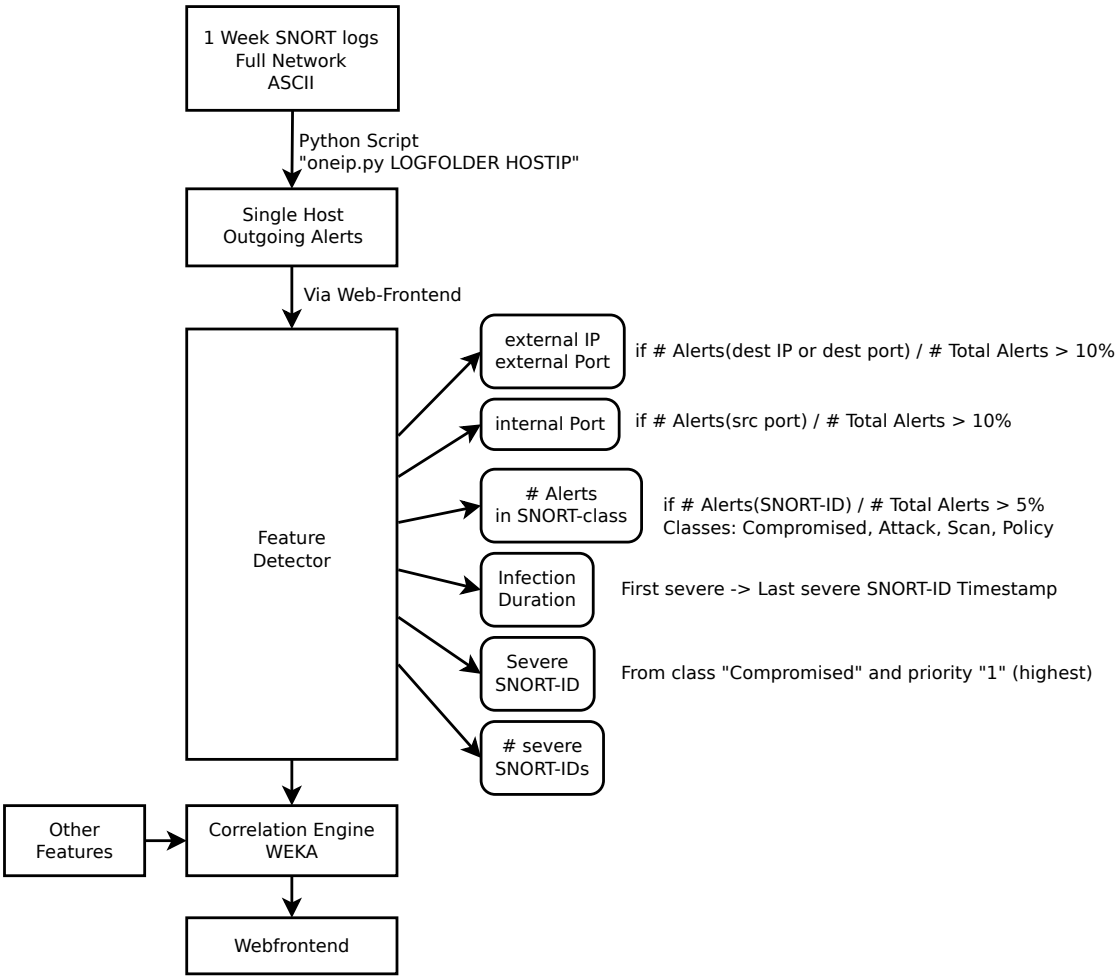


Figure 3.2: Process of extracting significant features from Snort logs

3.2.2 Vulnerability Scanner OpenVAS

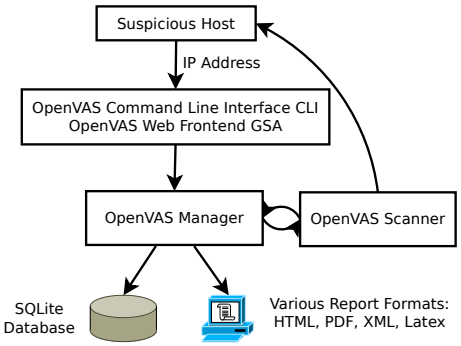


Figure 3.3: Overview of the vulnerability scanning process with OpenVAS

OpenVAS (Open Vulnerability Assessment Platform) [37] is a community-driven platform for vulnerability scanning. It was forked from the Nessus Vulnerability Scanner [38] when Nessus was turned from an open (GPL) software to a proprietary software in 2005. The most recent version (4.0) was published in March 2011 after a long period of inactivity. The project is backed by the BSI (German Federal Office for Information Security) and some commercial companies who are selling Network Vulnerability Tests (NVT).
We compared OpenVAS 4.0 to Nessus 4.0 on few suspicious nodes, using the freely available NVT-Feeds for both programs (for Nessus the feed is only for free for private use). In our tests the OpenVAS scanner found more vulnerabilities and also provided more useful insight into the vulnerabilities than Nessus did. For this reason we chose OpenVAS for further analysis.
OpenVAS can scan a whole network, a list of hosts or a single host. The most convenient form of interaction with the software is provided by the web interface *Greenbone Security Assistant* (GSA). After defining a scan policy by choosing a subset of tests from the NVTs, we scanned a specific suspicious host. After starting the scan job using the GSA web interface, OpenVAS Manager handles the execution of the task. A dedicated OpenVAS Scanner process performs the scan and reports the progress and results back to the OpenVAS Manager. The OpenVAS Manager can then provide various reporting formats,e .g. a HTML page or a XML output which can be easily used for further analysis steps. We used the detailed HTML report to analyze the security state of the investigated host. Figure 3.3 shows the general process in which a security assessment with OpenVAS is conducted.
The time needed for a security scan can vary a lot. It depends on the amount of reachable services and the number of tests available for these services. Scans are mostly non-intrusive, but some tests can cause reduced system performance or even a system crash. For this reason only manual non-intrusive scans outside of office hours were initiated. OpenVAS was not used for automatic scanning as part of the web framework.
A detailed example describing the gained knowledge from an OpenVAS report can be found in Appendix C.

3.2.3 Blacklists

Figure 3.4: Used blacklists: apews.org, dshield.org, emergingthreats.org, shadowserver.org and urlblacklist.com

Multiple sources for blacklists exist[5]. They serve three main purposes. Firewalls are using blacklists to block known attackers. These kind of blacklists can be retrieved from the blacklist provider in a format understood by the firewall. E-Mail anti spam systems use blacklists to block known spammers. Some of these blacklists can be downloaded, but most use the DNS system to query the blacklist. Content and access filters use blacklists to block the usage of services, for political, organizational or legal reasons. These blacklists are often based not only on IP addresses, but also on URLs.
We focus exclusively on downloadable lists. DNS lists are useful to block E-Mail spammers who are changing their IP addresses frequently. They do not offer historic data, which is used in our methodology. For the bulk processing of many IP addresses, which is often required in our approach, downloadable lists offer better performance. For future real-time use of the framework, an inclusion of good DNS based blacklists might be helpful.

The blacklists from 5 public blacklist providers (Figure 3.4) are automatically downloaded and stored on a daily basis. The blacklists are partly labeled by the providers, hinting at the reason for the blacklist entry. The most often used reasons are *Bot*, *Attacker*, *RBN* and *Spam*. If no reason is given or if the blacklist contains IP addresses from multiple sources, we used the reason *Multiple*. Urlblacklist.org provides additional tags related to the use of the website hosted on the IP address, e.g. *Socialnetworking* or *Webmail*.

For the analysis the 5 blacklists are combined. One or multiple days, depending on the period in which the infections occurred, are chosen from the stored blacklists. All IP addresses and reasons for their existence in the blacklist are extracted, again by using the Linux *grep* command for performance reasons. They are fed into a SQLite database, storing each IP address only once. The primary key in the corresponding database table is the IP address. Future searches for these IP addresses can be done with a time complexity O(log(n))[39].
A suspicious internal host is analyzed by extracting all contacted IP addresses which trigger a Snort alert. This is done similar to the process described in Section 3.2.1. Instead of extracting the Snort features, all outgoing alerts are analyzed and the external IP addresses are extracted. The SQLite database is searched for these IP addresses. If multiple hits are found, the results are aggregated. The results are stored in another SQLite database in textual form and can easily be displayed, e.g. on the webfrontend. The process is visualized in Figure 3.5

[5]For a idea about how many black lists exist, http://rbls.org/ is a good resource

```
ads: 1
multiple: 43
rbn: 2
chat: 1
```

Listing 3.2: Blacklist hits for outgoing connections of a host

Example

A host is suspicious because two of its connections end at IP addresses belonging to the RBN. Further blacklist hits are from ad-related servers and a chat server. In addition, many IP addresses belong to the *multiple* class. These hits mainly originate from DShield's unfiltered blacklist of hosts being blocked at one of the firewalls of their customers. DShield suggests not to use this data for blocking as they contain many false positives. We included this data in our set of features as it gives us a hint of possible malicious activity, but we did not trust this feature.
Based on this data alone it is not possible to infer if this host is infected. In this case, a scan of the host found a skype supernode. The RBN related alerts originated from this service and are a typical false positive case.

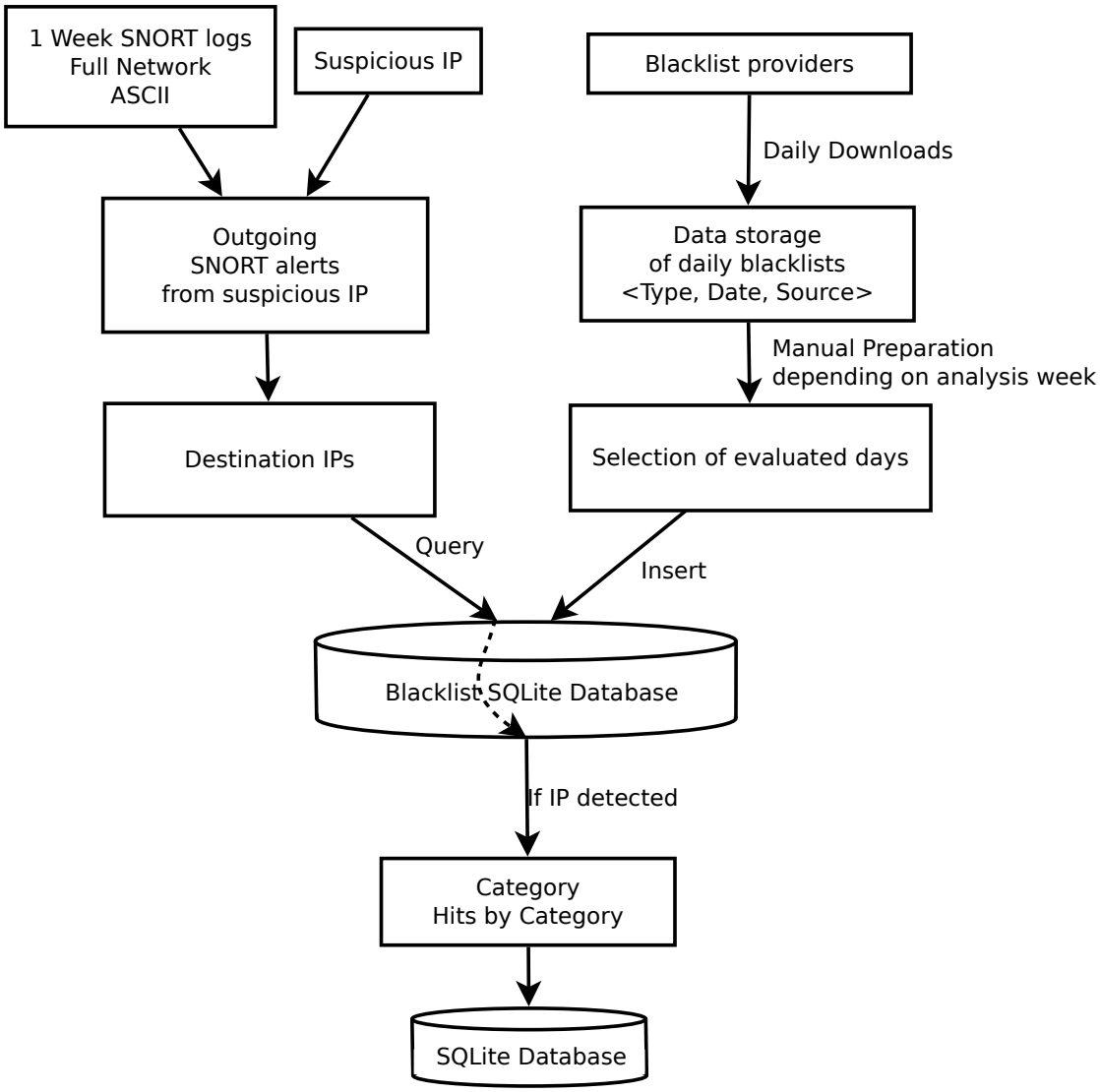


Figure 3.5: Process of checking for hits in blacklists

3.2.4 Nmap Network Mapper

Nmap network mapper is a popular open source tool to scan hosts for open ports. Originally it was developed to generate a map of the network. A large community developed many additional scan types and enhanced features. Important features of Nmap used in our scans are:

- Open/Closed/Filtered Port Detection
- SYN scan
- Connect scan
- Operating system detection
- Service version detection
- Top-port scanning which scans top-N most often used open ports

Only static hosts are scanned. Hosts with a dynamic IP address assignment come from VPN, public docking or WiFi connections. Since in this case the IP address mapping changes frequently it is quite likely to scan a different host than the original alert triggering host. Dynamic hosts are filtered out by reverse DNS lookups with the help of the Linux tool *dig* : the hostname usually contains the words “dynamic” or “public” or “nat” in this case.
At the beginning of a scan the Snort alerts generated by the suspicious host are analyzed. A list of source ports used in outgoing connections is generated. These ports and the 50 most often used open ports in the Internet are scanned on the suspicious host. This port list is the result of work done by Gordon Lyon, the original author of Nmap [40] see Appendix B.
The scan type for open ports is a *connect* scan. Additionally for each open port the service version identification is done. Nmap will try to open a full TCP connection consisting of an initial SYN packet, the SYN/ACK reply packet if the port is open and a final ACK packet from the scanner. If the port is not open, the host replies with a RST packet or does not reply at all if a firewall blocks the port. While this scan is less stealthy and slower than other scans, no root privileges are needed. The results of the scan are identical to the often used SYN scans. The service version detection uses a number of signatures and queries which it will send to the open port. The reply allows Nmap to determine what software version is running on this port.
After finishing the scan the results are collected and a list of reachable services is created. As a final step, a second run of Nmap tries to discover the operating system which is used on the host. Nmap uses OS fingerprinting [41] to guess the operating system. These results are not always accurate, as many devices have similar characteristics. Nmap tries not only to distinguish between Windows, Linux and Mac OS X, but also to find various network devices like routers, switches, printers and many more. To us only hits concerning Linux, Windows or Mac OS X are of interest, as the vast majority of systems being infected and discovered by Snort falls into one of these classes. Nmap has an accuracy score between 0 and 100. We only consider the most accurate operating system guess above an accuracy score of 80 if multiple systems were found. If the OS could be detected successfully, it is also saved in the SQLite database storing all other scan results.

Example

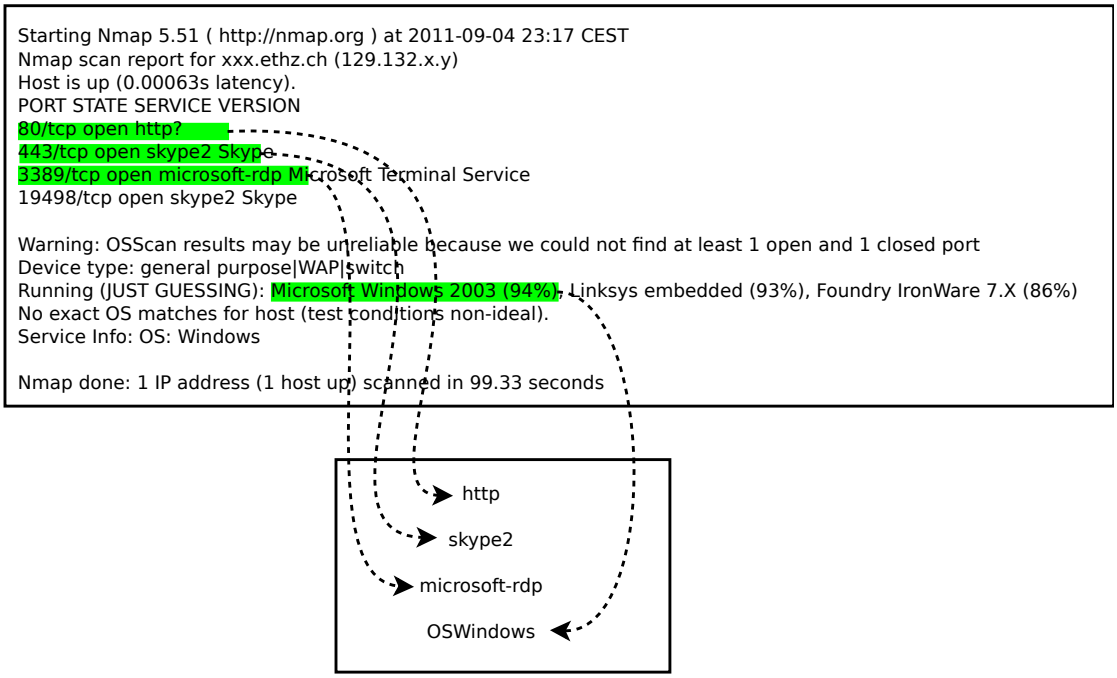


Figure 3.6: Example of a Nmap scan and the extracted features

The host from the blacklist example is now scanned using Nmap. Multiple source ports appeared in the Snort data and were extracted. After scanning these ports and the top 50 ports 4 open ports were found: port 80 (HTTP), port 443 (Skype2), port 3386 (Microsoft RDP) and port 19498 (Skype2). The detected OS was MS Windows. The Nmap output in a reduced form and the extracted features are displayed in Figure 3.6.
This example shows the advantages of the used scanning techniques. With a random port scan for e.g. the top 1000 ports the port 19498 would likely have been missed, as Skype chooses these ports randomly. Only by checking explicitly for ports found in Snort alerts we could find this open port. The correct classification of the service behind port 443 was done by Nmap's service and version discovery. Without this test the port would have been classified as a HTTPS port, because webservers with SSL encryption use this port normally. Skype uses this port, just as port 80, in its supernodes[6] because these ports are seldom blocked by firewalls.
The original reason for this host to be investigated were several alerts triggered by connections into the RBN. The connections originated from port 19498 are very likely replies to Skype connection queries from hosts in the RBN. This kind of traffic is not malicious and caused us to classify this suspicious activity as a common false positive.

[6] Skype supernodes are the backbone of the P2P system which handles the distribution of the lists of online users

3.2.5 DNS host name

The Domain Name System (DNS) translates from hostnames like www.example.com to an IP address like 192.168.0.1. If a reverse record for an IP address has been registered, DNS can also return the hostname if the IP address is known. If a query for "1.0.168.192.in-addr.arpa" using the reverse notation of the IP address is made, the reply will hold the hostname in the DNS PTR (Pointer Record) field.
We use this system to get additional information about the purpose of the suspicious internal host. In our network environment each IP address has an assigned hostname. From these hostnames we can easily find out if the IP address belongs to the dynamic address range if the name contains *dynamic* or *public*. Server hostnames often contain the service like *ftp*, *dns*, *mail* or simply *server*. When using unencrypted wireless and a landing page for authorization, no public IP address is given to the devices, but they are put behind a NAT[7]. The IP addresses of the NAT gateways can be identified because they have *nat* in their hostname. Of course these names can easily be changed or misleading, but in our tests not a single mislabeling occurred.
The process of extracting the possible purpose of the host starts with a reverse DNS query to our network DNS server. The retrieved hostname is searched for one of the keywords and, after finding a keyword, the tag is saved to the SQLite database. Possible tags are *Dynamic*, *Server*, *NATGateway* and *DNSServer*. The reason for the last tag are the frequent false positives originating from DNS servers, which we can quickly identify if we know that the investigated IP address belongs to a DNS server. A full list of keywords and tags can be found in Appendix E.

Example

The IP address 82.130.71.35 is suspicious because of several trojan related alerts. With "dig -x 82.130.71.35" the hostname of this IP address is retrieved: "guest-docking-nat-1-035.ethz.ch.". The keyword *nat* is detected and this host is tagged as a *Dynamic* host. This host will not be scanned by Nmap as we would only be able to scan the NAT gateway, not the client itself. In addition we know that this alert originates from an laptop connecting to the network with unencrypted wireless. Possibly multiple Hosts share this IP address, making it hard to establish which packets belonged to a single host. For this reason, we did not investigate these hosts further.

[7] Network Address Translation, a technique to connect multiple devices to the Internet over a single external IP address

3.2.6 Google Automatic Querying

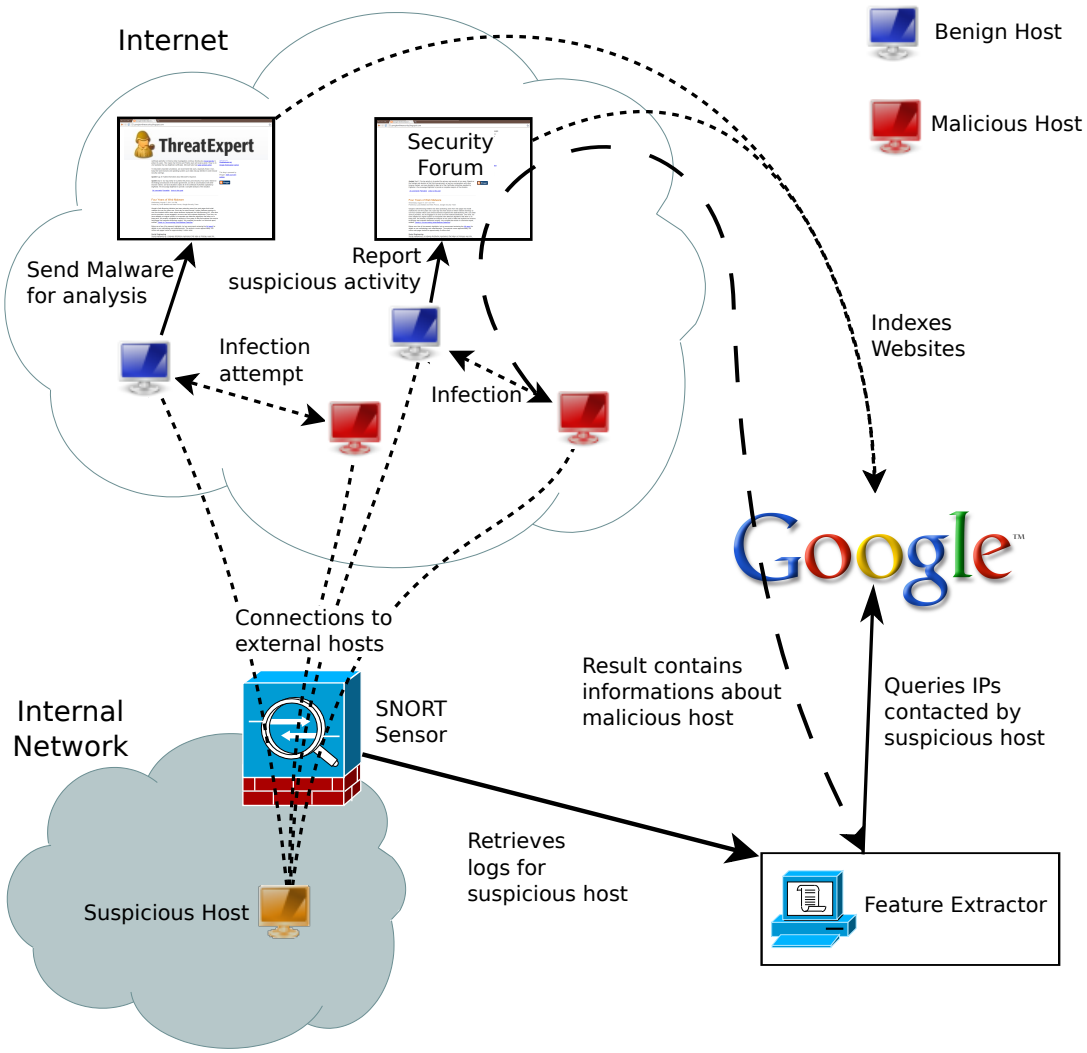


Figure 3.7: Information retrieval about malicious activity using Google Search Engine

Google's search engine is used to infer if IP addresses contacted by suspicious internal hosts are associated with malicious activity. This is done by querying google for the IP addresses and searching for keywords in the search results.

The idea behind this approach is the same as for blacklists: you are probably not the only one being targeted by the malware. If other parties have observed suspicious activity from certain IP addresses, they sometimes write about this in public forums. Malware analyzers like ThreatExpert [16] record outgoing connections. The contacted IP addresses can be found in their ThreatReports which are indexed by Google. Malware tracker pages like SpyEye Tracker [32] and Zeus Tracker [31] have lists of IP addresses being connected to these botnets[8]. These informations and possibly many more are found by Googles search engine. Compared to blacklists this information is often more detailed. The kind of malware connected to the IP address, the used IP ports and protocols (UDP or TCP) and the time when this infection was discovered can all give valuable insight into the nature of the discovered infection.

[8] A botnet is a collection of infected hosts which can be remotely controlled, e.g. to send Spam or attack Websites

In a first step, IP addresses are extracted from the Snort data. Only alerts of the classes *Attack* and *Compromised* are considered. The classes *Scan* and *Policy* are ignored as they refer to alerts which are not triggered by malware (*Policy*) or are not likely to contact malicious IP addresses (*Scan*). Furthermore a single alert type with more than 30 different contacted IP addresses is discarded. We can only query 100 IP addresses per day for free with one Google API key. Therefore above this threshold we inspect destination IP addresses connected to a single alert only manually. In most cases this kind of behavior originates from P2P network traffic like Skype or request-reply traffic as e.g. used for DNS.
For the Google searches their custom search API [42] is used. using the Python binding from the Python module *Google API Python Client*. The search results are stored in a PostgreSQL database for later analysis of the discovered words and to maximize the usage of the 100 free queries. A second search for the same IP address will not be served by Google, but by the PostgreSQL database. A query returns the 10 most relevant webpages as determined by Google. We did not collect additional search results. Most queries do not return a significant amount of pages. Almost 80% of the queries returning at least 1 result do not return more than 10 results. We assume that not many interesting information is lost by not considering the remaining search results and it enables us to query more IP addresses each day. Each query for 10 additional results would costs 1 of the 100 free queries.

The results are analyzed by searching for specific keywords. These keywords are searched for in the page name (e.g. “ThreatExpert Trojan Zeus”) and the text snippet provided by Google together with the page name. The keywords are taken from the paper *Unconstrained endpoint profiling (googling the internet)* [30] and augmented by our own security relevant keywords. Each keyword is associated with a tag. The most important keyword-tag combinations are (Bot|Trojan|Worm) - “Bot” and (malware|spyware|spybot) - “Malicious”. Additional tags are “Blacklisted”, “Adware” and “P2P”. The full list of keywords and tags can be found in Appendix E.
Once all tags have been found for an IP address, they are saved together with their frequency in a SQLite database. If malware or bot related keywords are found, the direct link to this search is saved in addition for quick manual inspection.

Keyword - tag combinations are established manually. This process can be done once and is quite robust as the important keywords do not change a lot over time. To establish a list of keywords, we used the search results from more than 3400 queries for IP addresses found in the process described above. A list of words with 3 or more letters and at least 10 occurrences was created, containing 1331 different words. As this process had to be done only once, we used these words to infer good keyword - tag combinations manually.

The whole system is summarized in Figure 3.7.

Example

A host generated several high priority Trojan related Snort alerts. A list of contacted IP addresses is generated and automatically queried. in all search results 8 times a keyword connected to the *Bot* tag is found and 21 keywords indicating *Malicious* activity are found. In total 11 IP addresses are either reported as being bot- or malware-related. A manual check of the search results confirms that the IP addresses are used to distribute malware for different types of bots. The specific alerts generated by Snort and the confirmation of connections to malicious IP addresses is a very strong indicator for an infected host.

3.3 Feature Correlation

A trained security engineer can often tell if a host is infected by just looking at a couple of selected features. In cases of doubt, he can try to get additional information. This process is based on his understanding of the way the malware operates. He also knows the reason for the alerts and can infer when a benign program did trigger an alert. Making an *automated* system perform the same task for all kind of malware is impossible.

Nevertheless, we are using automated systems to help the security engineer. The sensors which provide the features are often automated systems. Some of them can detect certain type of malware on their own if the detection rule is specific enough. But all systems are specialized on certain types of malware. No system can catch all kind of infections equally well.

In this section we will describe a learning algorithm helping the security engineer to infer if a host is infected. The algorithm uses results from previous infections. He compares the features of new suspicious cases to already classified detections. The reason triggering an automated classification must be reported by the system, such that the security engineer can better interpret the results. This way he can spot errors made by the algorithm and provide his own judgement. For this reason we do not consider algorithms where the decision process is not transparent. The two most prominent classifiers falling into this class are *support vector machines* and *neural networks*.

3.3.1 Classification Algorithms

Naive Bayesian Classifier

One of the simplest classification algorithm is the naive Bayesian classifier. It weighs all features as being equally important and independent from each other. It uses Bayes rule to compute the probabilities for a new case to belong into one of the classes. Lets assume we classified several alerts according to the data in the following table 3.1:

Snort alert	Operating System	Classification
Virus	Win	Infected
Spyware	Linux	Infected
Spyware	Linux	Infected
Virus	Linux	Clean
Spyware	Win	Clean
Virus	Win	Infected
Virus	Win	Clean

Table 3.1: Example of collected features and the corresponding classification

A new suspicious case is now found: Snort alert *Virus* and OS *Linux*. We calculate the likelihood of this case being also infected. A Snort virus alert is observed in 2 of 4 infections and the OS Linux is also observer in 2 of 4 infections. In total there are 4 infections in 7 cases. Multiplying these ratios gives us the likelihood for the class *infected*: $\frac{2}{4} \cdot \frac{2}{4} \cdot \frac{4}{7} = 0.143$. The same calculation for the *clean* class results in a likelihood of 0.041. A naive Bayesian classifier predicts that our new case is over 3 times more likely to be infected than to be clean.

Although the assumption of independence between features is an oversimplification, this

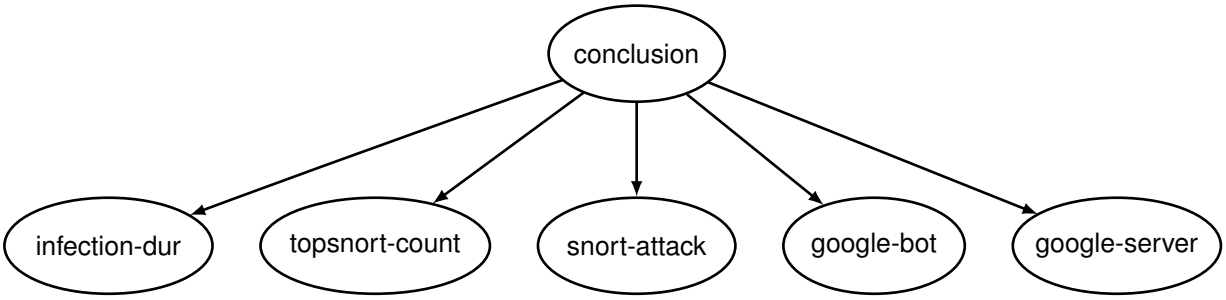


Figure 3.8: Naive Bayesian network extracted from security incidents from June 2011 - reduced to 5 features

method results in surprisingly good results on many real world examples, e.g. to filter E-Mail spam [43]. However naive Bayesian classifier are not suitable for our feature correlation. A rich training set is needed to infer accurate correlations between features and classes. Features which never appear together with a class have veto rights, making sure that this class never gets chosen when the feature appears in another test case. For example, if the feature *OS Linux* has never been combined with an *infected* case, the naive Bayesian classifier will never classify a new case as being infected if this feature is connected to the new case. Workarounds exist by assuming that every feature is found at least once for every class. This can introduce errors if the training set is not rich enough. We do not have a rich training set, and a single feature can often not be used for classification alone.

Bayesian classifiers can be visualized as Bayesian networks. The nodes represent features or classes and the edges are correlations between features. The value of each feature and class is given by a function with the parent nodes as inputs. For a naive Bayesian classifier, this graph has the class at its root and all features one level below the root. An example of a graphical representation of a naive Bayesian classifier is given in figure 3.8.

Tree augmented naive Bayesian classifiers

To overcome the limitations of naive Bayesian classifiers several solutions were suggested. A *tree augmented naive Bayesian network* (TAN), proposed by Friedman et al in 1997 [44], adds a simple structure of dependencies between the features to the naive Bayesian classifier. An edge is drawn from a feature to another feature, weighted with the conditional mutual information given the class. This complete graph of all features is then transformed to a tree by constructing the maximum weighted spanning tree. One can say that the result is a tree where features are connected only when the value of a feature tells a lot about the value of a neighboring feature. Finally all features are connected to the class, producing a graph as shown in figure 3.9.
A new case with features a_1 - a_n is evaluated by finding the class c which maximizes the posteriori probability $P_N\,(c|a_1, \ldots, a_n)$. N is the TAN learned from the training data.

The authors report that TANs outperform naive Bayesian networks and other state-of-the-art classification algorithms [44]. We found only very minor differences in both classification approaches for our data. The tree structure derived from the algorithm shows correlated features, but cannot be used without calculating the posteriori probability when a new case is found. The reason for a case being classified e.g. as *infected* is not visible from the tree structure. It can only be found by evaluating the probability functions at each node.

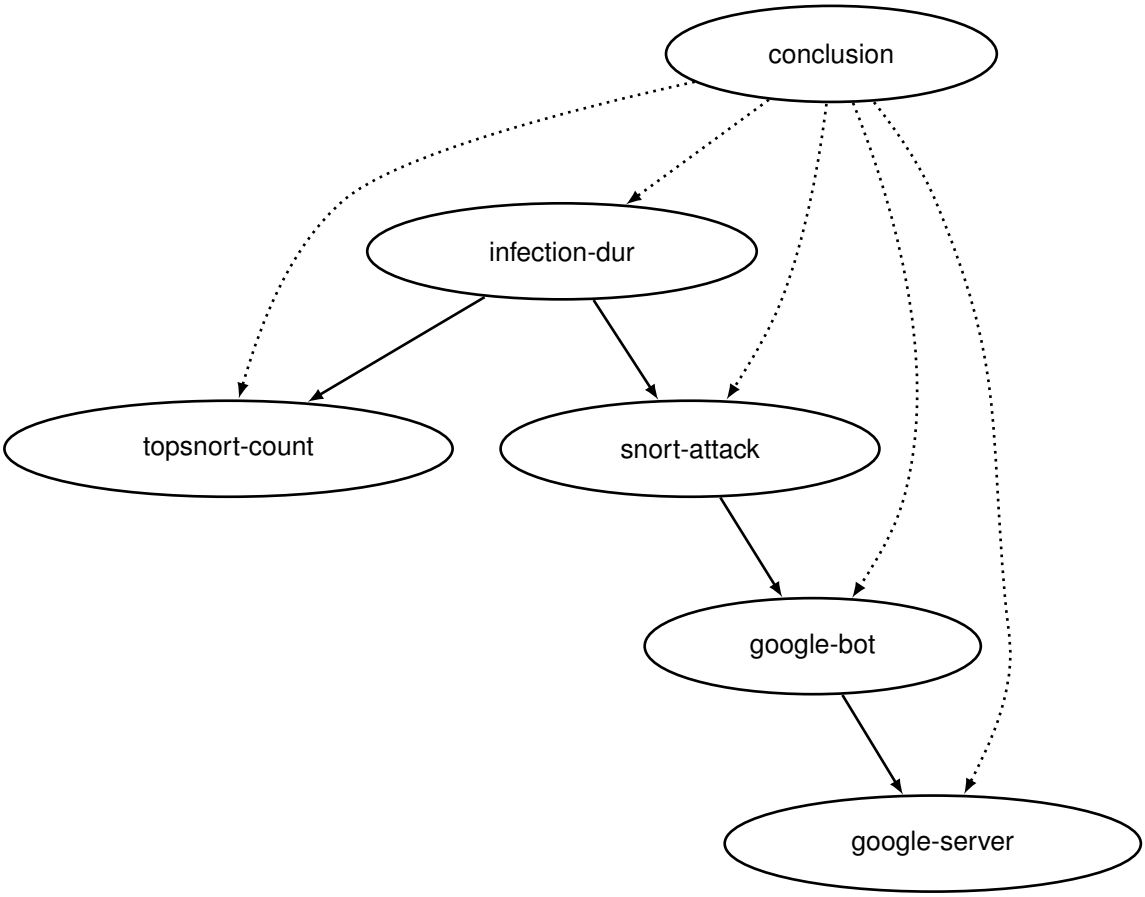


Figure 3.9: Tree augmented naive Bayesian network extracted from security incidents from June 2011 - reduced to 5 features

C4.5 decision tree classifier

C4.5 is an algorithm developed by Quinlan in 1993 [45] to create a decision tree. It can be used for classification by following the decision tree. The goal of the algorithm is to split the data at the feature which provides the highest information gain. The difference in entropy is used to find the best feature. The algorithm can be compared to human reasoning. For an efficient manual classification, it is best to first look at general features which can classify many cases. The algorithm mimics this behavior.

After choosing a feature which splits the data optimally, a node with this feature is created. The decision criteria is based on the possible values of the feature. If two or more discrete values are possible, e.g. *TRUE* or *FALSE*, the data is split on these values. If the feature is a continuous value, e.g. the count of Snort alerts, the algorithm finds the value which splits the data optimally regarding the entropy. This procedure is repeated for the remaining data, adding new nodes as children of the originally chosen feature. If only one class remains in the data, a leaf is created with this classes name. Future cases ending here after traversing the tree will be classified with the class name of this leaf.

Often the training data or the data in the new cases is incomplete. *C4.5* is tolerant against missing cases in the training data by only considering features which provide a high information gain. If data is missing in the new cases which should be classified, a missing value is interpreted as having all possible values. For each possible value the tree is evaluated and the class which is finally chosen most often is used for classification.

A further problem is that a complete decision tree can be very big and over-fitted to the training data. *C4.5* handles this problem by subtree replacement and subtree raising. Subtree replacement will replace a whole subtree with a leaf if most of the cases in that subtree are of the same class. Subtree raising will remove a feature with little information gain by one of its subtrees and incorporate the lost classes in the subtree's leafs. To establish when to use subtree replacement or subtree raising, an approximate error induced by this operation is calculated. If the error is lower after replacement or raising, the operation is performed.

The algorithm is used a lot in the industry [46] and most other classification algorithms are tested against this algorithm. A commercial version exists which is faster than the open source version. *C4.5* is implemented as part of Weka [47] under the name J48, referring to the last open version of *C4.5* Revision 8. We used this implementation to compute the decision trees. An example of a *C4.5* tree is given in figure 3.10

C4.5 was chosen to classify the security incidents. The high readability of the decision tree was the main criteria. A security engineer can easily follow the tree manually to classify new infections. New interesting correlations which can be used to create automatic rules can be extracted. In chapter 5 we will also show that the classification accuracy of *C4.5* is comparable to other tree-based classification approaches and outperforms the popular *Support Vector Machine* (SVM) classification algorithm.

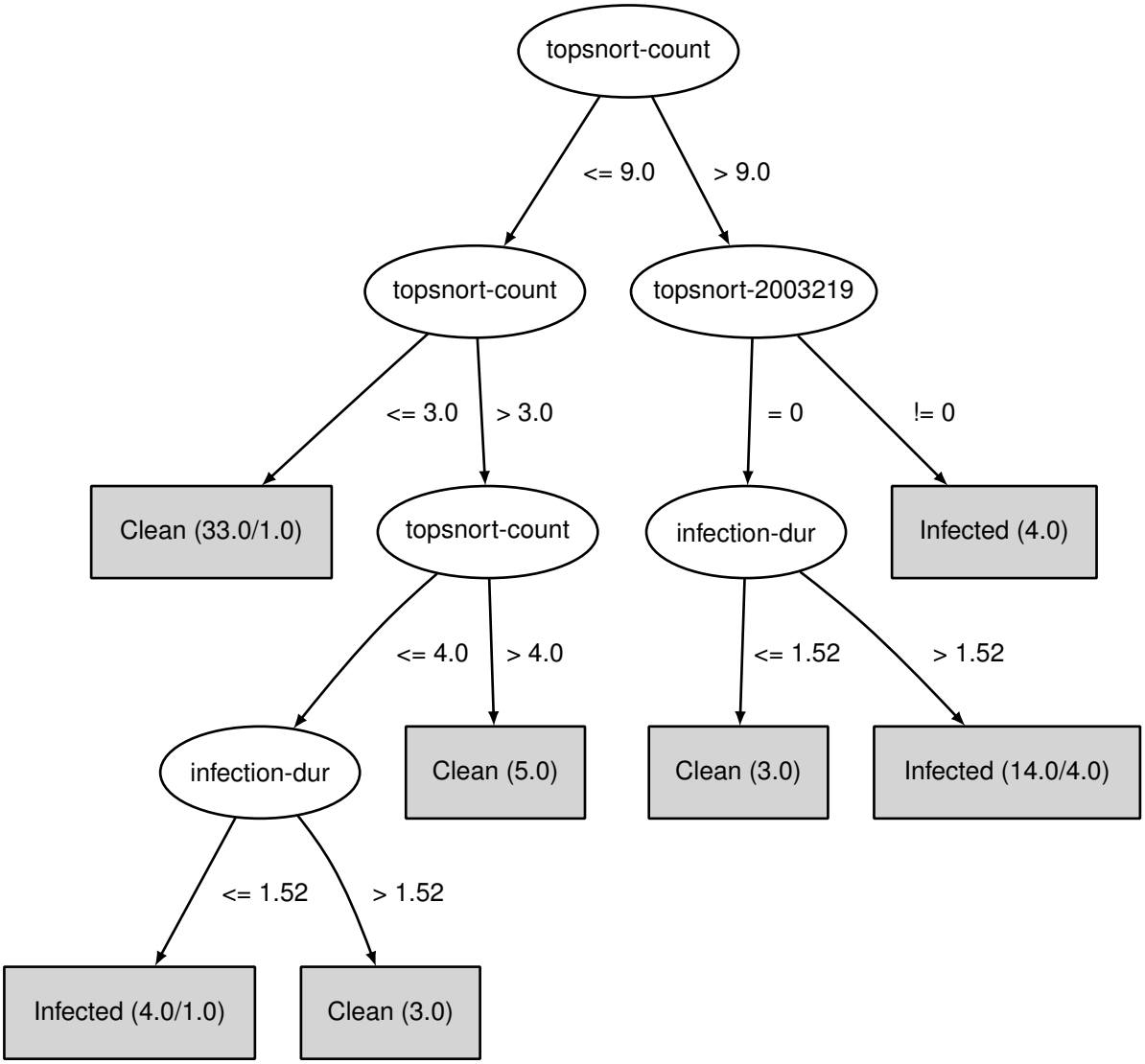


Figure 3.10: *C4.5* decision tree extracted from security incidents from June 2011 - reduced to 5 features

3.3.2 Implementation

In the following we used the Weka [47] suite to create the decision tree and classify new cases. It is an open-source data mining toolkit developed at the University of Waikato in New Zealand, written in Java. It features a graphical interface and a command-line interface. The command-line interface was used for automated generation of the tree and classification of new cases. The graphical interface enabled us to quickly test different algorithms and evaluate the multiple options available to modern classification algorithms.

Weka reads data in the Attribute-Relation File Format (ARFF) [48]. We converted the features we extracted from the various sensors into this format. All features are either binary, e.g. if a port is open or closed, nominal e.g. the DNS name which can be put into several groups like *dynamic*, *NAT* or *Server*) or continuous e.g. the number of Snort alerts triggered by the suspicious host. Some features need to be combined to reduce the total amount of features. This is done for Snort alerts where not the Snort id is used as a feature, but the total amount of hits in a Snort class (*Policy*, *Attack*, *Compromised* or *Scan*). The time of an attack is transformed from the start - end notation to a duration in hours.

All successfully classified incidents are then used to create a decision tree with Weka's J48 implementation of the *C4.5* algorithm. The tree is exported to the DOT-format for a graphical representation. With the help of the Linux command line tool *dot* the textual DOT format is transformed into a PNG image which can be viewed with many image viewers and web browsers.
If a new case is evaluated, a classification is suggested based on a decision tree generated from all previously classified incidents.

An overview of the process is given in figure 3.11.

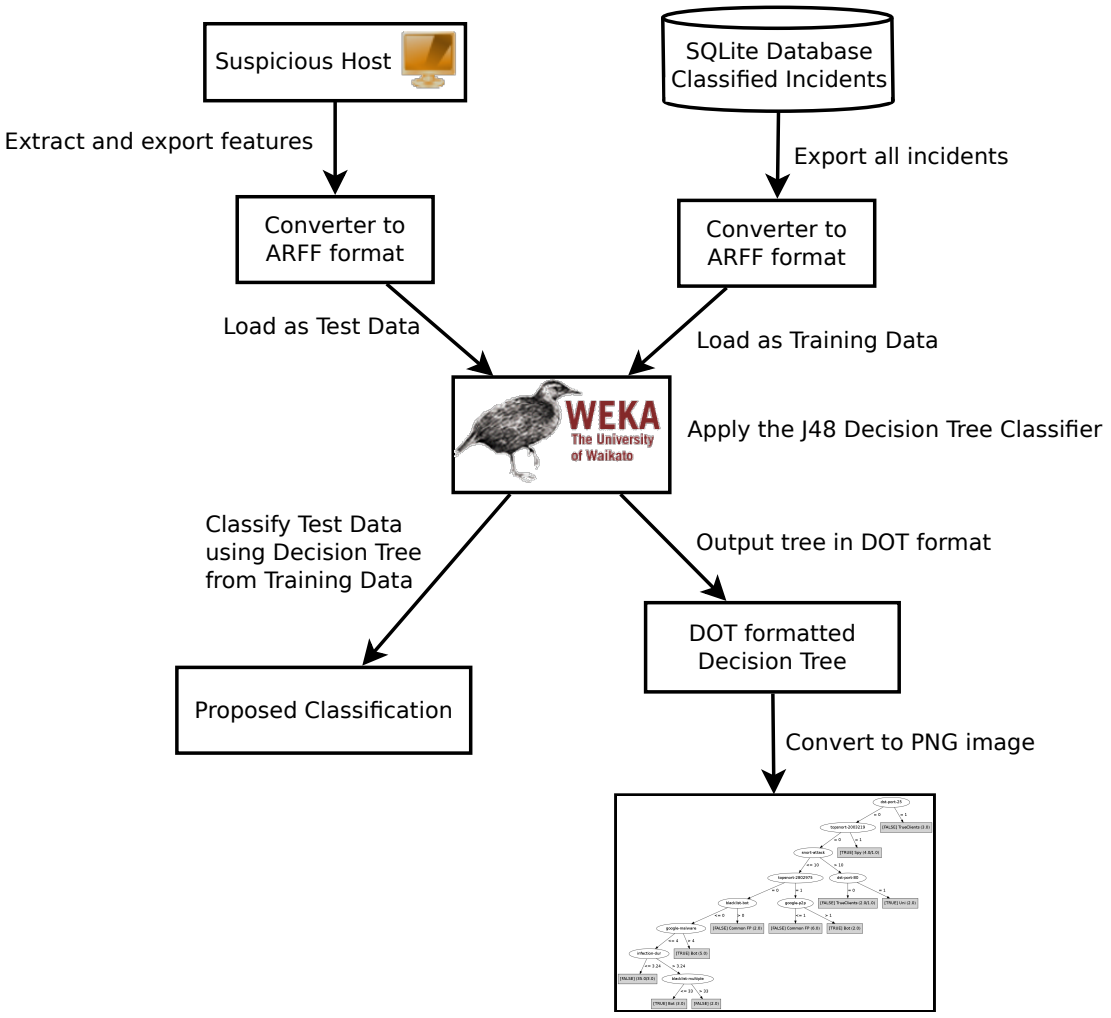


Figure 3.11: Process of classification using Weka and the J48 classification algorithm

Chapter 4

Host State Assessment

Host state assessment is the combination of the feature extraction and classification approach. For new threats this task has to be done manually. Known attacks can be extracted into rules and used for automatic classification. Our framework aims at giving the security administrator as much information about a host as quick as possible. A classification based on previous incidents is proposed, but the decision is left to the security engineer. The decision tree which is extracted automatically from the classified incidents is displayed to guide the manual classification. The implementation of the back end and front end are explained in the following sections 4.1 and 4.2.

4.1 Back end with Python and Flask

The feature extraction and feature correlation is controlled with the help of Python scripts, utilizing the micro-framework Flask [49]. Flask is a Python library which makes it very easy to quickly create interactive websites. It features a powerful debugger allowing to debug the code on demand directly from within the web browser. A screen shot showing the interactive console used for debugging is displayed in figure 4.1.
The three tasks of the back end are *feature extraction and correlation*, *accepting user input* and *providing information* to be displayed at the front end.

4.1.1 External Modules

Whenever possible, existing Python libraries were used to provide a functionality. Important external modules which were used are:

- *subprocess* and *os* to call Linux command line programs.
- *jinja2*, *werkzeug* and *flask* [49] to create the website and react to HTTP events.
- *psycopg2* and *sqlite3* to connect to PostgreSQL and SQLite databases.
- *apiclient* [42] to connect the Google's Custom Search API.
- *silk* [50] to handle IP address manipulation.
- *nmap* [51] for a direct interface to the network mapper nmap for automated host scans.

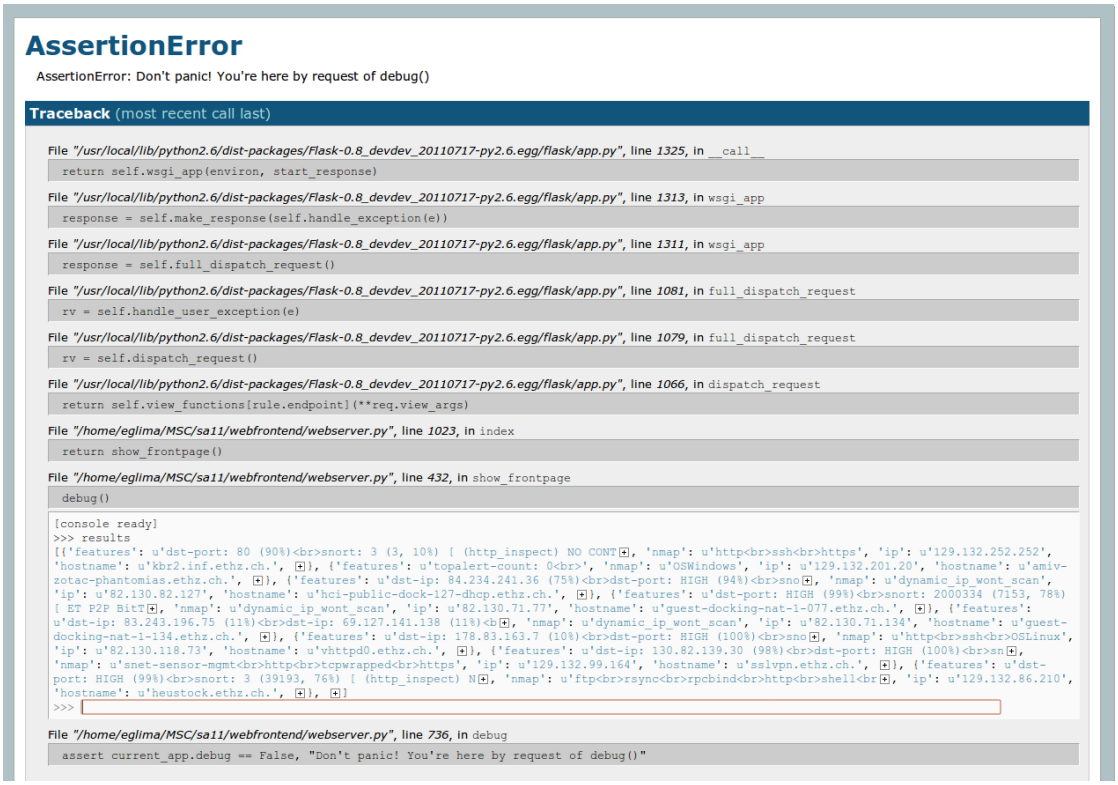


Figure 4.1: Debugger view of the embedded web server in Flask

4.1.2 Architecture

The central part of the back end is a light weight web server powered by *Flask* [49]. It receives HTTP requests and calls the functions chosen to answer to these requests. Content is delivered to the web browser by the template engine Jinja2 [52].

The back end receives the suspicious IP address and a compressed list of Snort alerts from the front end. Flask saves the compressed Snort alerts on the hard drive such that the file can be loaded at a later time without need to upload it again. The IP address and the filename of the Snort alert file is stored as a session variable. The front end starts the feature extraction and correlation by sending HTTP GET requests calling the functions, e.g. a HTTP GET to the URL /nmap will cause the network scan against the suspicious IP address to be started. The back end provides additional information needed by the function. The back end extracts the Snort alerts from the compressed Snort alert file and passes them on to the feature extraction.

The feature extraction functions will save their results in a SQLite database. This SQLite database is configured as part of the Flask initialization and can be accessed by any function being called from the back end. There is no need to create multiple connections to this database. Access to the database is handled by Flask.

The feature correlation is called whenever a new incident should be classified or when a new conclusion has been added to an incident. The feature correlation retrieves all classified incidents and the previously extracted features for the new case from the database. The generated decision tree is saved on the hard drive and passed on to the web front end for visualization. The proposed classification for the new incident is returned. Flask uses this information to propose a classification to the security engineer using the web front end.

The process is visualized in figure 4.2.

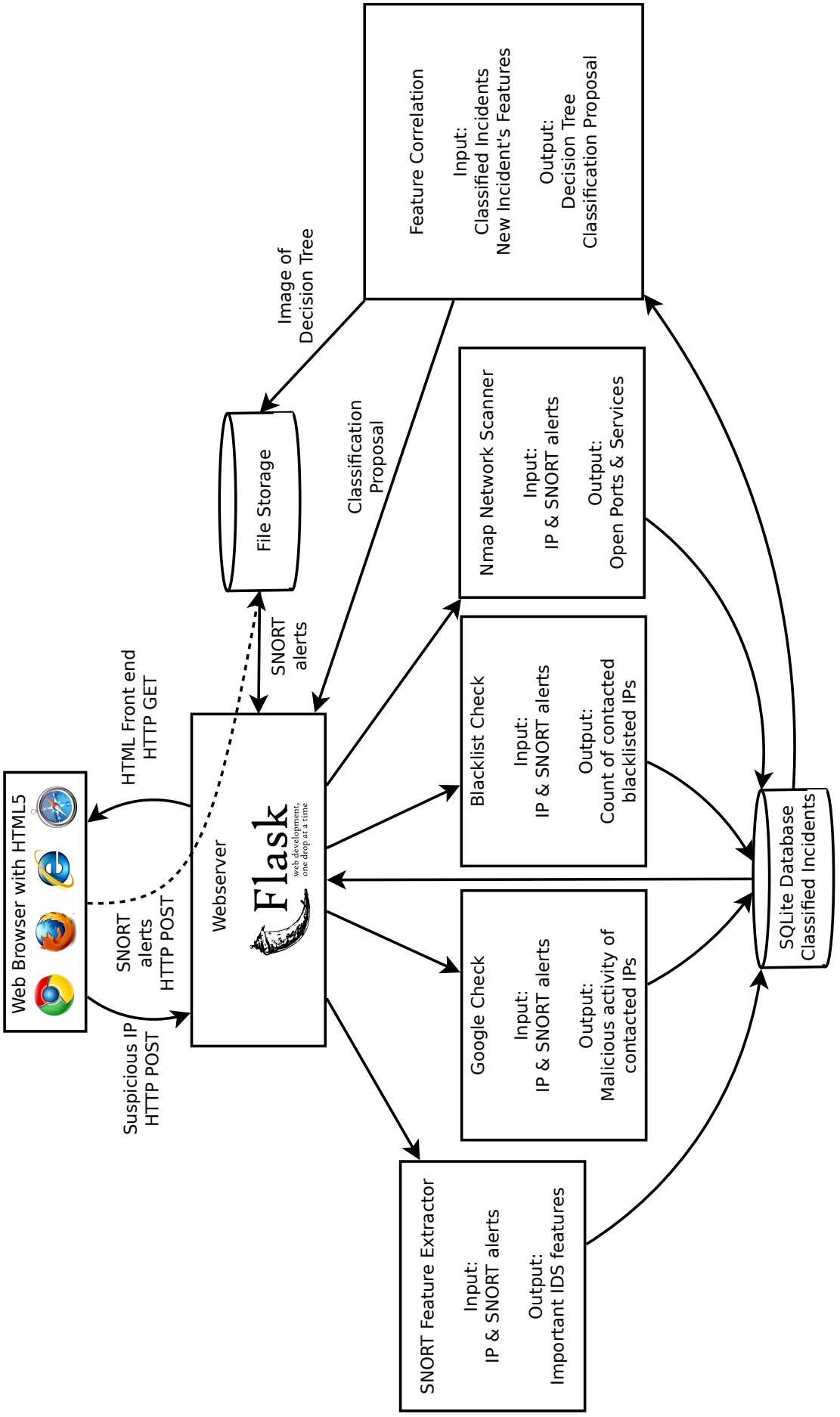


Figure 4.2: Overview of the architecture employed in the back end

4.2 Web front end

The primary way to interact with all feature extractors and the correlation engine is done by using a web front end. The front end can be used on any HTML5 capable web browser. Using the main page a new incident can be investigated by providing the suspicious IP address and the corresponding Snort file. The Snort file must be in Snort's textual output format and should only contain outgoing alerts generated by the suspicious IP address. All other alerts are discarded. The file must be compressed using the gzip algorithm.
The main page also displays the decision tree and the already classified incidents. Old incidents can be loaded and re-evaluated. A screen shot of the front page is given in figure 4.3.
The feature extraction is started as soon as a new case is created by providing both the suspicious IP address and the Snort file. The security engineer can ask the back end to extract the features by clicking the appropriate link. If contacted IP addresses should be analyzed by Google, a list of contacted IP addresses together with their alerts is displayed. The search engine can be chosen. Either the *normal* version searching on all pages or the *malware* version only searching malware-related web sites can be selected. In addition, the IP addresses must be chosen which should be checked by the search engine.
A screen shot of the feature extraction and the search engine interface is given in figure 4.4 and 4.5

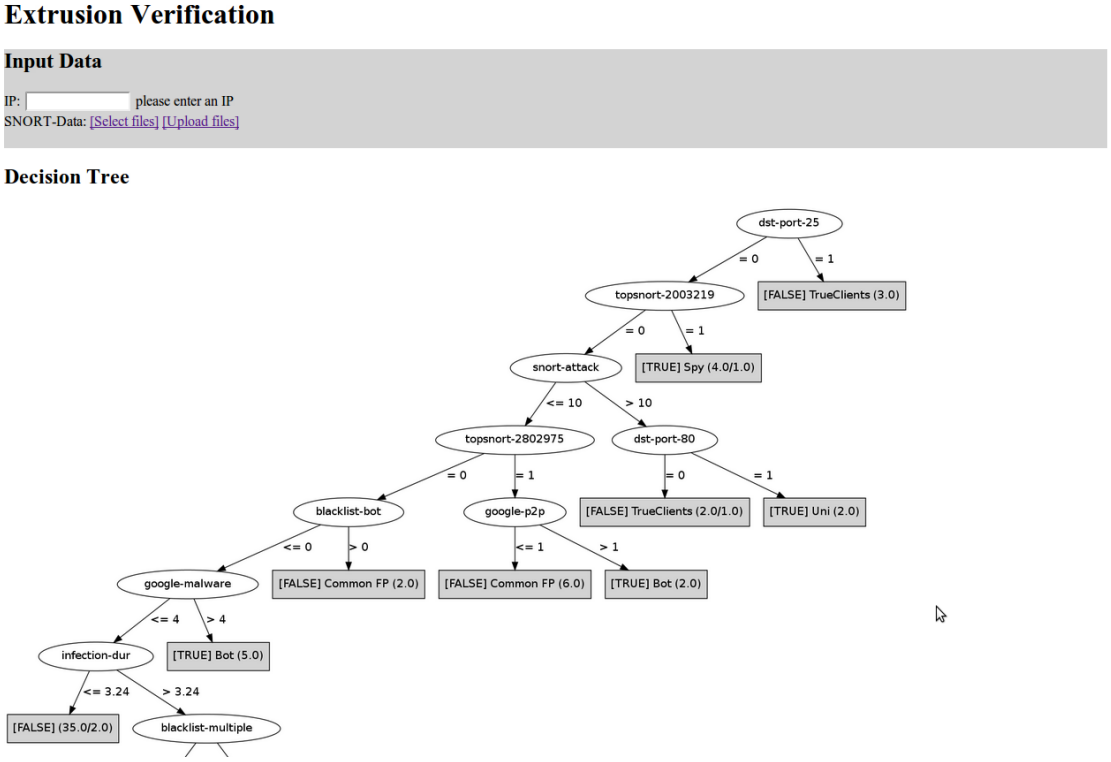


Figure 4.3: Screen shot of the web front ends main page, showing the form to create a new case and the decision tree

Extrusion Verification

Input Data

IP: 129.132.
Snort-File: 129.132. -analyse_this.log.gz

Check new IP

Process Data

Reverse DNS
Find specific Snort Alert features (Ports, IPs, Alert Type Distribution)
Classify node (nmap)
Find Blacklisted IPS (for all remote IPs)
Google IPs with normal/malware search (choose alert class or IPs)

Results

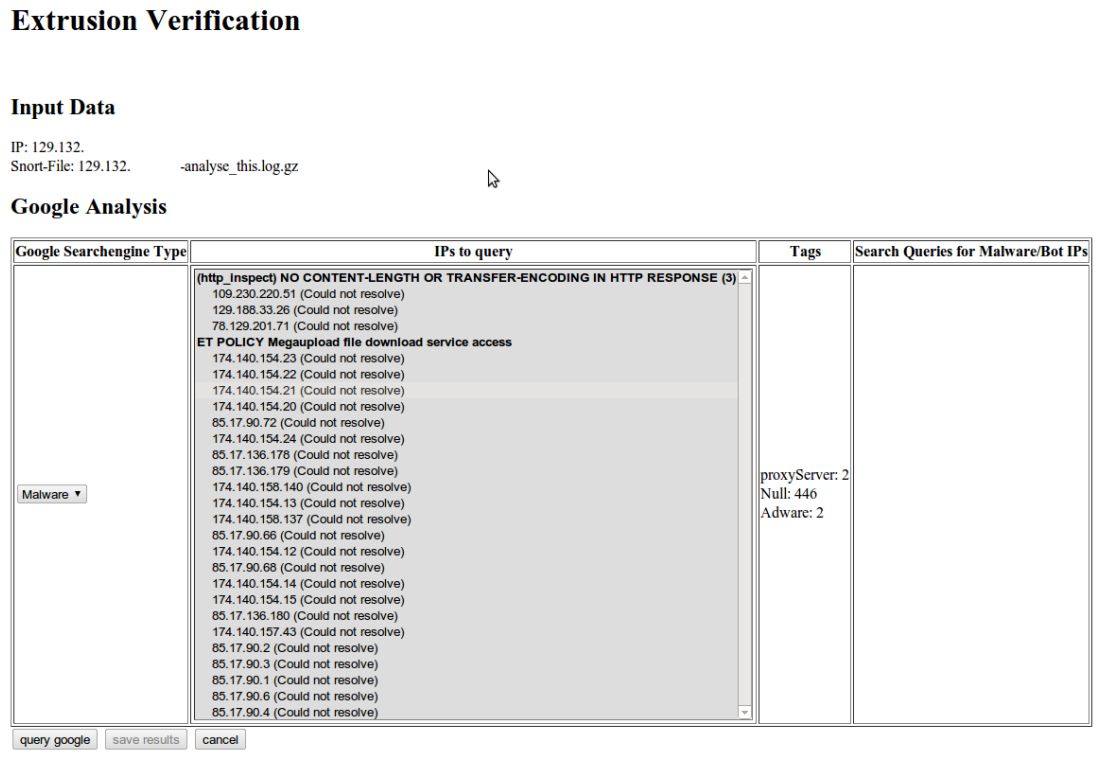
IP	Snort-File	Hostname	Scan Results	Blacklist	Features	Google Tags	Google Queries	Conclusion
129.132.	129.132. analyse_this.log.gz	.ethz.ch.	http ssh https	no hits	dst-port: 80 (90%) snort: 3 (3, 10%) [(http_inspect) NO CONTENT-LENGTH OR TRANSFER-ENCODING IN HTTP RESPONSE] snort: 2009301 (27, 90%) [ET POLICY Megaupload file download service access] src-port: HIGH (90%) topalert-count: 0			[FALSE] *Proposed: [FALSE]*

Conclusion

Conclusion Missing
[TRUE] Bot
[FALSE]
[FALSE] Common FP
[TRUE] Uni
[?]
[TRUE] Spy
[FALSE] TrueClients

New Reason: (Press Enter to Confirm)
Save conclusion & check new IP

Figure 4.4: Screen shot of the web front ends feature extraction page. All features for the investigated IP are already extracted.



Extrusion Verification

Input Data

IP: 129.132.
Snort-File: 129.132. -analyse_this.log.gz

Google Analysis

Google Searchengine Type	IPs to query	Tags	Search Queries for Malware/Bot IPs
Malware	(http_inspect) NO CONTENT-LENGTH OR TRANSFER-ENCODING IN HTTP RESPONSE (3) 109.230.220.51 (Could not resolve) 129.188.33.26 (Could not resolve) 78.129.201.71 (Could not resolve) ET POLICY Megaupload file download service access 174.140.154.23 (Could not resolve) 174.140.154.22 (Could not resolve) 174.140.154.21 (Could not resolve) 174.140.154.20 (Could not resolve) 85.17.90.72 (Could not resolve) 174.140.154.24 (Could not resolve) 85.17.136.178 (Could not resolve) 85.17.136.179 (Could not resolve) 174.140.158.140 (Could not resolve) 174.140.154.13 (Could not resolve) 174.140.158.137 (Could not resolve) 85.17.90.66 (Could not resolve) 174.140.154.12 (Could not resolve) 85.17.90.68 (Could not resolve) 174.140.154.14 (Could not resolve) 174.140.154.15 (Could not resolve) 85.17.136.180 (Could not resolve) 174.140.157.43 (Could not resolve) 85.17.90.2 (Could not resolve) 85.17.90.3 (Could not resolve) 85.17.90.1 (Could not resolve) 85.17.90.6 (Could not resolve) 85.17.90.4 (Could not resolve)	proxyServer: 2 Null: 446 Adware: 2	

query google | save results | cancel

Figure 4.5: Screen shot of the web front ends search engine interface.

Chapter 5

Results

In this chapter we will present the identified security incidents at the ETH campus during two periods in sumer 2011. An overview of the methods which have been used to reach a conclusion is given. The impact of the different features is evaluated. Finally the chosen automated classification algorithm C4.5 is evaluated and compared to other classifiers.

5.1 Overview

Period April - May 2011

The first analyzed period spans from 01.04.2011 to 12.05.2011, covering 42 days. In total 49'587'064 alerts where generated, from which 27'896'155 (56%) were caused by internal hosts. 382'491 unique external hosts and 22'644 unique internal hosts generated alerts. We investigated 235 unique internal hosts which exhibited more serious signs of an infection. The hosts were selected by the method developed by Raftopoulos in 2011 [12].

From these 235 unique hosts 149 (63%) belonged to the dynamic address range which is used mainly by unmanaged personal devices from students and employees.

Period June 2011

The second analyzed period spans one week in June, from 20.06.2011 to 27.06.2011. This week was pointed out to us by the IT support group because of a very high number of unusual alerts during that time. Even though only 7 days were analyzed, in total 121'402'841 alerts where generated, from which 34'781'873 (29%) were caused by internal hosts. 239'863 unique external hosts generated alerts and 17'742 unique internal hosts generated alerts. We investigated 96 unique internal hosts which exhibited more serious signs of an infection. The hosts were again selected by the method developed by Raftopoulos in 2011 [12]. In addition, 30 hosts which were not reported by this method were investigated. The purpose was to give examples of unbiased behavior of non-infected hosts for the classification algorithm.

From the 126 investigated hosts, 61 (48%) belonged to the dynamic address range.

5.2 Security incidents

The first period from April until May 2011 was analyzed with the various tools described in chapter 3. During that process, the tools were tuned and optimized. We focused on the different kind of malware and the features which enable us to classify the infections correctly. The web front end was developed after the first period and used to classify all incidents from the second period, i.e.the special week in June 2011. The second period was also used for automated alert classification.

From the 235 unique hosts which were evaluated in the first period, 43 hosts were discarded because they belonged to the range of addresses assigned to NAT devices. NAT devices serve as access points to the Internet for multiple clients. As such, it is very hard to infer which client caused an alert. Combination of multiple alerts under the assumption that they origin from the same host is not possible. As our inference is based on multiple alerts and not one single incident alone, we discarded these hosts.

The biggest share of successfully classified incidents from the first period belonged to the *spyware* class. 118 hosts were classified as having installed spyware. In the second period only 18 hosts were infected with spyware. With few exceptions, the reason for the classification was, that the hosts had installed the *Ask Search Toolbar* [53]. This software is classified as malware by SNORT as this toolbar frequently gets installed without the user being aware that it is installed, e.g. when installing other free software which bundles this toolbar. The reason for the few detected spyware cases in the second period is likely due to the shorter period in which we searched for infections.
Detecting the *Ask Search Toolbar* is done by SNORT with specific rules. The toolbar identifies itself as *AskTB* or similar names in the User-Agent string of a HTTP request. This behavior is very specific and has not been observed for any other software. As such it is not needed to use other sensors to infer if this toolbar has been installed.
For the analysis of more serious malware we discarded the hosts exhibiting only spyware behavior and those behind NAT devices. In total, 74 incidents remained from the first period. From the second period, 108 incidents remained. These totals serve as a basis for calculating the percentages of the different types of infections. For the evaluation of the importance and relative weight of different features, only the incidents from the first period were used.

Class	Cases	Percentage
Unknown	25	14%
Clean	67	37%
Common False Positive	37	21%
Infected Clients	20	11%
Bot Infected	31	17%

Table 5.1: Types of identified security incidents in period April-May 2011 and in one week in June 2011

5.2.1 Incident Classes

We used in total five classes to classify our incidents:

- *Bot Infected* labels incidents where a Trojan was detected.
- *Infected Clients* refers to those cases where the host is a server, which itself is not infected, but which is used by infected clients. Those clients abuse the server, e.g. as a mail server, a DNS server or a web proxy server. They cause the server to exhibit suspicious activity, e.g. by causing the DNS server to query domains known to serve malicious content.
- *Common False Positives* refers to hosts which were believed to be infected because of seemingly malicious activity exhibited by benign applications. Skype and a antivirus solution are examples for this behavior, but also some SNORT rules which trigger wrongly on benign traffic.
- *Clean* marks cases were we inferred that no infection occurred.
- *Unknown* belongs to those hosts where we could not infer if an infection occurred. In most cases too few alerts generated by SNORT caused this inference, although the alerts were severe. An example is a single known malicious query by a Trojan with no further activity at all. While the host may well be infected, we could not find any information about the contacted IP address or observe other malicious activity. In few cases the alerts and signs did show non-consistent behavior. Known malicious traffic together with contact with the RBN occurred, but the malicious traffic was directed at benign sites, e.g. google.com or facebook.com.

In table 5.2.1 the number of incidents per class from both periods is summarized.

5.2.2 Bot Infections

During the first period of 42 days 13 infections were found which exhibited consistent bot-related activities. In contrast in the second one week period 18 bot-related infections were found. A wide variety of bots was detected. With 4 hits, the *Conficker* Bot was found most frequently, followed by *Polybot* with 3 hits. *Blackenergy*, *Mufanom*, *SpyEye*, *Nervos*, *Ransky*, *Torpig* and *Kryptic* were discovered once in different hosts. For all these bots specific SNORT signatures exist which were use to infer which bot infected the host. For 17 bot infections the kind of bot is not known. For these incidents the consistent malicious activity caused us to classify them as bots. In most cases multiple contacts into the RBN occurred. Other alerts which are marked as *possibly malicious* by SNORT did contact hosts which were tagged as malicious by the search engine approach.

5.2.3 Common False Positives

In 37 cases a host did show seemingly malicious activity while not being infected, but executing a program which tends to trigger security alerts. As malware is constantly changing, security sensors try to watch out for generic malicious behavior patterns instead of specific malicious activity. Some benign software also tends to act like malicious software, e.g. because it is scanning a wide range of addresses in a network. Other benign software is not following the standards or good practices, e.g. by giving a wrong User-Agent string when browsing the web.
In some cases, the SNORT alert rule is not specific enough and triggers on normal traffic. These rules are normally discovered quickly by the community and removed from the ruleset. While they are active, they trigger some alerts which we also classified as common false positives because it is easy to detect them based on their SNORT rule ID.
In particular the following programs and Snort rules caused false positives:

Skype

Skype operates a P2P network of so-called super nodes. Every Skype user with a direct connection to the Internet, i.e without a firewall or NAT device in between, can become a super node. These super nodes are used to by normal clients to find out if their contacts are online and to circumvent firewalls and NAT devices. They have connections to many other hosts in the network. Some of these hosts are on the RBN and ShadowServer blacklists embedded into SNORT. Consequently, they trigger many alerts of traffic which originates from the super node and goes to a malicious network or host. The traffic itself is not malicious in these cases. Overall 10 cases in which a Skype super node caused false alerts were detected.

ICQ WebChat

ICQ [1] features a chat functionality which can be accessed using their web page chat.icq.com. The used protocol is the popular Internet Relay Chat (IRC) protocol. However IRC is also known to be a communication channel for bots. SNORT watches for traffic which looks like IRC traffic, but is not directed at the official IRC ports 6660-6669. As these ports are often blocked in a corporate environment, bots use other ports to connect to their controllers. ICQ's web chat is using port 80 and as such triggers alarms of IRC traffic on non-IRC ports.

Avira AntiVir

The popular free virus scanner from Avira does use an incorrect User Agent string when connecting to the avira-servers during an update. SNORT warns when this string is found because malware also uses the same string.

SNORT rules

Two rules used by SNORT were found to be causing alerts with few other malicious activity at the suspicious hosts. The SNORT rule with ID 2007626 is used to find traffic from the Trojan *Pitbull*, but frequently triggers on benign traffic. The rule has been deleted in the meantime. Another SNORT rule from a commercial rule set watches for queries to a Chinese web page which tracks user activity. Even when visiting benign Chinese pages which use this tracking

[1] ICQ (a homophone for "I Seek You") was the first widely used Instant Messenger

service, this rule is triggered. Some new malware is also using this tracker to provide the bot controller with the addresses of infected hosts. While this rule likely triggers on true infections, we discovered that most of the time no further suspicious activity occurred at the host.

5.2.4 Infected Clients behind Servers

DNS, proxy and mail servers often seem to be transmitting malicious traffic. A DNS server which is used as a caching name server from hosts in the internal net will contact IP addresses in malicious networks if its clients try to resolve hostnames from these networks. A mail server receives mails from clients in the internal network and forwards them to mail servers in the Internet. These E-Mails sometimes carry sensitive data or are being sent to known malicious mail servers. A proxy server is used to cache and filter web traffic. It also hides the true address of a user, making traffic seem to originate from the server instead of the client. If malicious web sites are contacted by a client using a proxy server, SNORT only detects the traffic from the proxy server.
In all these cases, the server itself is very likely not to be infected. The clients behind the server, on the other hand, are likely infected. We did not have access to the log files from the servers and could not infer the true origin of the alerts. We regarded these alerts as false positives as the server which was reported as being infected is not infected. For a security administrator with access to the server log files, these alerts can be important when trying to find the infected clients.
In total 20 hosts were classified as such servers.

5.3 Method Evaluation

The 4 different main methods used to infer if an infection took place are based on the *IDS SNORT*, the *search engine Google*, the *network scanner Nmap* and various *blacklists*. In addition, we used the *hostname by reverse DNS*. All methods contributed to the analysis done for the second period in June. For the incidents from the first period we evaluated which method was crucial to infer if an infection did or did not take place. We discarded all spyware related incidents. They can be detected based on SNORT alone and do not give interesting insight into the malware landscape as they do not try to hide, attack or redistribute themselves.
Overall, 74 such incidents were investigated. In table 5.2 the numer of occurences is given that a certain method was necessary for a successful inference. In 13 cases (18%) we could not determine if the host is infected even though all available methods were used. Figure 5.1 shows the combined methods and the percentage of cases categorized by the combination.

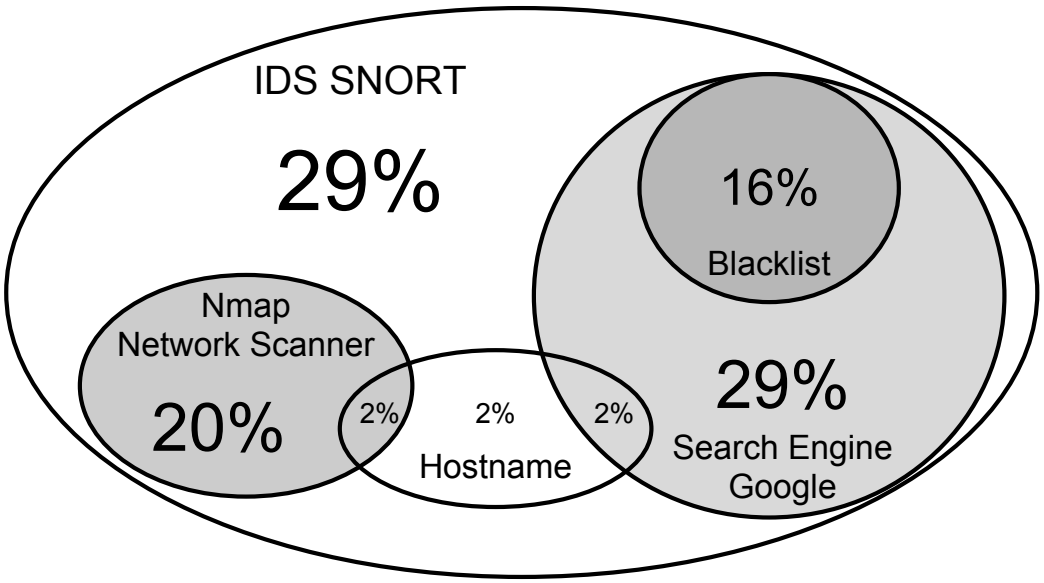


Figure 5.1: Combined methods and the percentage of suspicious cases categorized by the combination

Table 5.2: Number of times a security feature extraction method was necessary to infer if an infection occurred - taken from 74 incidents analyzed in April and May 2011

Method	Cases involving method	Percentage
IDS SNORT	61	82%
Search Engine Google	28	38%
Network Scanner Nmap	13	18%
Blacklists	10	14%
Hostname (reverse DNS)	3	4%

5.3.1 IDS SNORT

As SNORT alerts were the initial reason for inspecting a certain host, they also committed the most valuable data to our evaluation process. They were needed in every case as they provided the biggest insight into the activities of the host. Contacted external hosts were extracted from the SNORT data and used for the search engine and blacklist methods. When using Nmap to scan the suspicious host, SNORT provided an extended list of ports to scan. These ports were used by the suspicious host as source ports for potentially malicious traffic.
Beside being able to classify all 118 spyware incidents from the first period, inspecting SNORT alerts alone was enough to classify 18 infections. For the remaining 43 successfully classified incidents, it was necessary to combine other methods together with SNORT.

5.3.2 Search Engine Google

Using the search engine Google proved to be a successful strategy to check for malicious activity exhibited by contacted hosts. In 16 cases the combination of SNORT alerts and the search engine results alone enabled us to classify an infection. In 10 other cases it was necessary to combine the information with the information from the blacklist method. In two special cases a manual research was needed. These cases turned out to be rare false positives caused by a browser based game and a wrongly listed host from the search results. Overall, the method involving the search engine contributed to 28 successful classification (38%).
The importance of this method is also emphasized by the fact that only 3 malware infections found in the period in June did not show any signs of malicious activity when querying Google.
While malicious activity is detected in 15 of the 18 infections in the second period, it was also detected in 42 of 77 cases where we found no infection. The set of analyzed hosts is biased towards hosts which do show behavior which is at least suspicious, which partly explains the high number of false positives. It also shows that using a search engine alone to infer if a host is infected is not a good strategy.

5.3.3 Network Scanner Nmap

The network scanner was used to determine the operating system and open ports of the suspicious host in the internal network. This data was useful to filter out false positives and find servers which show malicious activity because of their clients. In 7 cases it was possible to detect a Skype super node with the help of a network scan. The hosts were then classified as *common false positives*. In 6 cases a network scan helped to discover that either a mail server, proxy server or DNS server was deployed on the host. The hosts were consequently classified as having *infected clients*, but not being infected themselves.
Overall, network scans filtered out 13 false positives.

5.3.4 Blacklists

Blacklists were used in 10 cases. In all these cases, they served as a sign of malicious activity together with the results from the search engine. As blacklist data is often reported as being outdated or inaccurate, we did not consider a host which contacts hosts on a blacklist as a sufficient feature. However when the host contacted many blacklisted hosts which are also reported to be malicious by the search engine, we concluded that an infection occurred. If both the search engine and the blacklists showed no hits, we concluded that the host is not infected. In 6 cases, based on this pattern, we found an infection. In 4 cases the host was marked as clean because no blacklist showed any hits.
Blacklists are still very useful tools to find malicious activity. When a host on a blacklist is contacted by an internal host, the reason should be found. Many suspicious hosts were found because SNORT utilizes two blacklists, one with hosts from the RBN and one with hosts controlling malware, found by the ShadowServer project. 13 of the 18 infected hosts found in period in June also contacted several hosts which are on one blacklist. Only 21 of the 77 non-infected hosts of the second period did not show a malware or bot-related hit on any blacklist. Almost all hosts contacted at least one host which is on one of the blacklists.

5.4 Feature correlation and classification

During the second period, the correlation and classification algorithms described in section 3.3 were evaluated. The classification algorithms Naive Bayes, TAN and SVM were used to compare against the accuracy of the decision tree algorithm C4.5.

To find a good measure of the accuracy of the different methods, we used a 10-fold stratified cross-correlation. In this method the training set is randomly ordered and split into 10 parts, also called *folds*, of equal size. The set is not purely random, but slightly modified ("stratified") to create the same class distribution as in the complete training set. After splitting the data, one part is used for testing and the 9 remaining parts are used for training. This is repeated for every fold of the training data. The results are averaged over all test runs and this gives us the cross-validation estimate of the accuracy.

5.4.1 Extracted Features

The decision trees were extracted from 114 classified cases of security incidents. The distribution of the 5 classes of infections is given in table 5.4.1. In total 113 different features were extracted.

Class	Cases	Percentage
Clean	44	39%
Common False Positive	20	18%
Infected Clients	13	11%
Spyware Infected	19	17%
Bot Infected	18	16%

Table 5.3: Types of identified security incidents in period June 2011

Most of these features represented a single type of SNORT alert with high severity. 46 different kinds of these SNORT alerts were observed in our training set. 29 of these alerts occurred only in a single incident. Nevertheless, these alerts are very important for the manual analysis of an security incident. Alerts which appear in multiple cases are also important for the automated analysis. All other SNORT alerts were counted by their class, resulting in 4 features called *snort-scan*, *snort-policy* , *snort-compromised* and *snort-attack*.
30 different services, including the *no-open-ports*- and *hostdown*-"services", were discovered to run on the suspicious hosts by the network scanner. 3 different operating systems were extracted: Windows, Linux and in one case iOS, the operating system used on mobile devices from the company Apple.
5 different kind of blacklists, including the feature *blacklist-nohits* when no entry on any blacklist was found, were used. The *RBN*, *malware* and *bot* blacklist represent specific blacklists which we believed to be more reliable to find malware than other blacklists. All other hits on any blacklist were combined in the feature *blacklist-multiple*.
6 distinct privileged source and destination ports were extracted. They belonged in most cases to web- (TCP 80 and TCP 443), DNS- (UDP 53) and E-Mail-Traffic (TCP 25). In few cases, SSH- (TCP 22) and FTP-Traffic (TCP 21) were discovered. All unprivileged ports were combined in the features *src-port-HIGH* and *dst-port-HIGH*.
The search engine method resulted in 5 different features. They corresponded to the security relevant context in which the external hosts were found. This context was grouped into *blacklist*,

bot, *malware*, *P2P* or *server* clusters. The keywords used to find the corresponding context are explained in Appendix E.
Based on the hostname a *dns-tag* feature was extracted, as described in section 3.2.5. The *infection duration* was measured in hours and exported as a feature. The number of important SNORT alerts generated by a host was the final feature used to automatically classify the suspicious hosts.

A list of all features can be found in Appendix D.

5.4.2 C4.5 Decision Tree

The main parameter which can be tuned when extracting decision trees from training data is the confidence. This value must be between 0 and 1, where 0 corresponds to no confidence in the training data and 1 is equivalent to full confidence. With full confidence, a huge tree is generated because the pruning is based on the believed rate of error in the training data. The lower the confidence, the more pruning of the tree is done. As the default setting which balances the accuracy and size of the tree, 0.25 is used by the Weka data mining tool set.
Another important factor is the minimum number of cases which must end at a leaf. Because of the high number of diverse attributes relative to the size of the training set (113 features for 114 incidents), it is not difficult to construct a specific tree with many leafs which correctly classifies all training data. This extensive tree is hard to evaluate by the security engineer and vulnerable to over-fitting on unimportant features.
To find the optimal parameters for the J48 algorithm, the parameter selection tool from the Weka tool set was used. The search range for an optimal minimum number of cases per leaf was from 2 to 5. The confidence value was searched in 0.02 steps from 0.01 to 0.91. We used the number of correctly classified cases when using stratified 10-fold cross-correlation as the evaluation function.
The resulting optimal parameters for our decision tree were a minimum number of 2 cases per leaf and a confidence value of 0.076. If values above this threshold are chosen, the size of the tree does not change, but the accuracy drops a little bit. Even for a high confidence of 0.5 and a minimum number of 4 cases per leaf, the difference is small: 80 cases get classified correctly instead of 83 cases when using low confidence values.

With these values the decision tree displayed in figure 5.2 was found.

Figure 5.2: Decision tree generated from 114 classified incidents from one week in June 2011 - confidence level 0.076

a	b	c	d	e	<– classified as
12	0	0	2	4	a = Bot Infected
0	13	0	0	0	b = Infected Clients
1	0	17	1	0	c = Common False Positive
0	0	0	18	1	d = Spyware Infected
1	0	0	4	40	e = Clean

Table 5.4: Confusion matrix of classified hosts - Based on the J48 implementation of the C4.5 algorithm - Training and testing on the same dataset

	True Positive Rate	False Positive Rate
Bot Infected	0.39	0.07
Infected Clients	0.92	0
Common False Positives	0.79	0.05
Spyware Infected	0.68	0.11
Clean	0.73	0.09

Table 5.5: True and False Positive Rate for the C4.5 decision tree

The decision tree starts with the elimination *servers hiding infected clients*. Mail servers can reliably be detected by checking for traffic going to port 25 (used by the Simple Mail Transfer Protocol SMTP). Interestingly a high number of contacts to hosts in the RBN is also a sign for such a server. An infected node does contact only few nodes in the RBN, if at all, while these servers have many contacts. A better feature to find these infections would be the source or destination port referring to DNS (port 53) traffic. When the blacklist feature is disabled, the C4.5 algorithm does indeed use these ports to find these cases.
In a next step, all *common false positives* are detected. Here the relevant specific SNORT rules are used as a main indicator. SNORT rule 2007626, 200347 and 2802975 frequently trigger on benign traffic. Only in 3 cases did we classify these hosts as infected. In these cases, we saw a high number of contacts to other malicious hosts. The decision tree uses the P2P feature discovered by the Google search engine to discriminate these hosts. This is consistent with our observations of hosts which do use possibly illegal file sharing activities and consequently get infected more frequently.
After all common false positives and servers hiding infected clients are filtered, the truly infected nodes have to be differentiated from the clean nodes. The decision tree first detects reliable SNORT signatures. In our data set, the rules with ID 2009024 (Trojan Conficker), 2012939 (Trojan Generic trojan) and 2005320 (Trojan Suspicious User-Agent MyAgent) could be used to classify 9 infections. The hosts which were classified as being clean did exhibit very few SNORT alerts of the class *compromised*. At this point in the decision tree we have the highest number of wrongly classified true infections. 4 bot- and 1 spyware-infected host were wrongly classified as being clean. 2 other bot-infected hosts were classified as being spyware-infected.
Some spyware is detected based on a specific SNORT rule (ET 2003219 - Alexa Spyware), but most spyware is found by checking for web traffic (TCP port 80) and the absence of any other alert.

A table of the confusion matrix showing the correctly and wrongly classified incidents when testing on the training data can be found in 5.4. The rate of true positives and false positives, determined by 10-fold stratified cross-correlation, is given in table 5.5.

	C4.5	TAN	Naive Bayes	SVM
Bot Infected	0.07	0.06	0.05	0.03
Infected Clients	0	0.01	0.03	0
Common False Positives	0.05	0.04	0.11	0.04
Spyware Infected	0.11	0.08	0.13	0.1
Clean	0.09	0.16	0.09	0.2

Table 5.6: False Positive Rates for several classification algorithms based on 10-fold stratified cross-correlation - security incidents from June 2011

	C4.5	TAN	Naive Bayes	SVM
Bot Infected	0.39	0.5	0.39	0.44
Infected Clients	0.92	0.92	0.92	0.92
Common False Positives	0.79	0.74	0.74	0.90
Spyware Infected	0.68	0.58	0.58	0.58
Clean	0.73	0.84	0.76	0.8

Table 5.7: True Positive Rates for several classification algorithms based on 10-fold stratified cross-correlation - security incidents from June 2011

5.4.3 Comparison to other classification algorithms

Only the decision tree algorithm gives us easily interpretable reasons for the classification of the suspicious cases. Other algorithms can be more precise in classification. We compared the C4.5 algorithm to a simple naive Bayesian classifier and a tree-augmented naive (TAN) Bayesian classifier. The TAN algorithm could classify one more case (73.7% correctly classified cases) than the C4.5 classifier, the naive Bayesian classifier did classify 5 cases less (68.4%) than the C4.5 algorithm. As the TAN algorithm only very slightly outperforms the C4.5 algorithm, we did not consider using it for the classification of new cases.

A Support Vector Machine (SVM) is an algorithms which represent the state of the art of modern classification algorithms. Many data mining and machine learning algorithms are based on SVMs and very good results could be achieved with these classifiers [54]. For this reason we used an implementation of a SVM in Weka to classify our results and compare it to the C4.5 classifier.
SVMs are, just as C4.5 and the Bayesian trees, supervised machine learning algorithms. A labeled training set is needed to create a model. Based on this model, the classification can be done. SVMs can be configured with many parameters, but a popular implementation of SVMs does handle this problem for us: Sequential Minimal Optimization by Platt from 1999 [55]. Weka has an implementation called SMO. We used the implementation to train a SVM. The performance was identical to the performance of the TAN algorithm and as such only slightly better than our C4.5 algorithm.

An overview of the true- and false-positive rates from the different algorithms is given in table 5.7 and 5.6

Chapter 6

Discussion and Future Work

6.1 Discussion

6.1.1 Security Incidents

The malware landscape is highly diverse. We found 9 different types of bots on hosts in the campus network. No bot dominated the number of infections. In more than half of the cases, no specific alert but the general behavior of the host caused us to infer that the host is infected. These results highlight the importance of a holistic approach to malware detection. Multiple sensors are needed in more than 70% of all cases when the trivial cases of easily detectable spyware are excluded. The decision tree uses features from SNORT, Google and blacklists. While testing the best setting for the confidence of the decision tree, we found that a tree generated with a slightly higher confidence in the extracted features uses all features, including the results from network scans and the hostname analysis. The great variety of the used methods confirms our assumptions that a multi-sensor approach is superior to single-sensor methods.

The second major finding is the high confidence achieved by automated classification for two types of apparently infected hosts. Servers which are not infected but have possibly infected clients and applications known to produce many false alarms are detected reliably. As these types of false positives will likely appear very often when searching for malicious hosts in a network, it is important to filter them quickly. Our approach represents a reliable method to detect these incidents automatically. Approximately one quarter of the detected cases in the week in June fell into this class.

From all methods we used, the IDS SNORT is the most important one. It provides data about the behavior of suspicious hosts. Analyzing the NetFlow log files can give a complete list of contacted hosts, but this list is likely to be very large and will contain many addresses which trigger an alarm in blacklists and our search engine method. It is necessary to combine all three information sources to achieve reliable results.
The network scan and hostname analysis are valuable tools to filter out false positives. SNORT generates millions of alerts in our environment. Even after preprocessing these alerts to find suspicious cases with the method proposed by Raftopolous, most alerts are not referring to true infections. Quickly filtering the common false positives in a first step of an analysis is vital to reduce the number of incidents upon which an administrator must act.

The search engine method together with blacklist checks provides a method to sort the remaining cases by importance. In many cases we could not find enough signatures of an infection. This does not necessary mean that the host is not infected, but it is less likely. Also these infections are likely to be less severe as they show no signs of attacking other hosts. In cases where SNORT gives a warning about possible infections because of suspicious traffic, e.g. by misspelled User-Agent Strings, additional hits in blacklists and through the search engine help to confirm the alert.

14% of all cases could not be classified with one of the methods. The number strongly depends on the applied security thresholds, which therefore determine the desired security enforcement level in a network. For the public part of the campus network in this environment, our threshold is a reasonable compromise. As long as a possible infected host does not start attacking other hosts and degrade the network performance while doing that, there is little incentive from the IT support group to act. Informing the user of suspicious activity by E-Mail is likely to be enough. Only in cases where an infection is detected in the managed environment or in an infrastructure-critical part of the network, the security engineer has to act and manually inspect the host. In the two periods of 48 days in total we found 31 bot infections, less than one infection per day. This number is well manageable compared to the hosts generating an alert (more than 22'000) and even to the number of hosts we inspected (361 hosts, 7.5 hosts per day).

6.1.2 Automated Analysis

The decision tree algorithm C4.5 proved to be well suited for an automated analysis. The way this tree extracts the relevant information is comparable to the cognitive approach of the security engineer. Some features are surprising, e.g. the choice to classify all nodes with more than 6 contacts to the RBN network as servers with infected clients. For this reason, manual inspection of the tree is needed to infer reasonable rules for a possible automated classification.
None of the other methods compared to C4.5 showed significant better results. On the other hand, even the very basic naive Bayesian classifier did not classify the incidents much worse than our classifier. As we have many different features and, compared to the number of features, only few cases in our training set, this result is not surprising.
The SVM classifier showed a better true- and false-positive rate for most classes compared to the C4.5 algorithm. Only for one important figure, the number of false positives for the class *clean* was significantly higher when using the SVM classifier. In case of doubt, the SVM algorithm seems to favor labeling a host as clean.

6.2 Future work

The results obtained with our method are based on the network traffic from the suspicious hosts. No access to the suspicious hosts was possible. As such, our results are missing this valuable source of information. Checking our *bot infected* and *clean* host with virus scanners might give us important additional information and could verify our assumptions.
Additional sensors based on NetFlow data can give interesting additional information. They are independent from the SNORT IDS data and can act as the source to cross check the data for the other methods. Anomaly detection based on NetFlow data is an active research field with promising new methods.
Our framework did focus on the classification of incidents extracted from SNORT data. Future work should investigate if the methods can also be applied successfully when a suspicious host is detected through other methods. In these cases fewer IDS data may be available.
The framework is not used for fully automated classification. A method of exporting the newly derived information into a Security Incident Management Systems should be implemented for a future productive use.

Chapter 7

Conclusion

In this thesis we showed the need of combining multiple methods to infer if a host is infected or clean. With our automated method of feature extraction, a host can quickly be assessed. Known attack patterns are detected by a decision tree algorithm and can easily be converted into fully automated patterns detecting future attacks of this type. We showed that this method is especially useful to filter out false positives. False positives are a huge problem with IDS systems, and our framework can help to solve this.
New attacks can be assessed by combining the information extracted from IDS logs and active scans of the suspicious host with information from the Internet. This way it is possible to classify new attacks quickly without having to change the detection framework.

While developing and testing our method, we built a labeled data set of infected nodes spanning 48 days. This set contains the timespan in which hosts were infected, either with a Trojan or spyware, and reasons for different kind of suspicious, but benign hosts. For this period we had access to the SNORT alert and NetFlow data of all hosts in our medium sized university network. This dataset can be used for future evaluation of new methods to find infected hosts in a network. It is not based on artificial traffic or a testbed. As such, it provides unique possibilities to test new algorithms on realistic data.

Appendix A

Original Task

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Security assessment of infection incidents in the ETH university campus

Master Thesis

Matthias Egli

Project Description

In recent years, malware infections have become one of the most prominent security threats in the wild. They provide the means for cybercriminals to steal confidential data, launch various types of attacks, host fraudulent web content, and send massive spam. As security practitioners shift their attention to this emerging threat and introduce defensive measures to safeguard their infrastructures, the attackers develop and deploy the next generation of trojans, clickbots, and spyware that have stealthy behavior and robust communication architecture making it very hard to monitor and detect them.

In our infrastructure, the ETH campus network, intrusion detection is used as the main line of network-based defense against active attacks and malware infections. In a daily basis the ETH security team collects and stores more than 3 million alerts which get triggered when a potentially malicious activity is manifested. Performing forensic analysis based on this data, i.e. identify infection evidence from large amounts of system monitoring traces, has become an important and very challenging task.

The main goal of this work is to provide forensics evidence of high confidence regarding active infections that are tagged as suspicious by peripheral defenses and detection engines. For this purpose we will use large amounts of monitoring data including IDS logs, NetFlow data, and DNS logs, online security threat reports and blacklists, combined with network probing and vulnerability assessment tools. We aim to correlate the output of these diverse security sensors in order to detect complex bahavioral patterns that are manifested by infected nodes.

We have to cope with two inherent difficulties of network security analysis. Firstly, the types of malicious behaviors that we attempt to identify are quite disparate, ranging from multi-stage attacks and worm propagation events to trojan and malware communication patterns. Secondly, the tools that we use provide a reasonable detection accuracy only at very fine levels of analysis (e.g. packet level signatures, server activity logs, DNS requests). Although, the produced observations might be related to malicious behaviors, these tools do not provide concrete evidence that the actual security incident has occured.

Our approach will rely on a rigorous off-line automated process, that combines multiple independent trace evidence of low confidence and our expertise in analyzing such incidents, in order to strengthen the assertion that the actual infection has occured. The sources of low or medium level observations that we exploit in order to derive high confidence evidence regarding the existence or absence of an active infection detected during the analyzed tracing period, are the following:

- Security tickets generated by our IT security team
- Independent blacklist data listing known malicious hosts
- Threat reports related to analyzed security threats
- Reconstructed episodes of multi-stage attack incidents
- Publicly available security profiling information
- Security assessment data collected using active scans

The second goal of this thesis will be to provide a visualization tool that will operate as a dashboard of forensics evidence related to active infections. It will provide the security operator with all the information required in order to investigate suspicious nodes in an efficient way, by incorporating the aformentioned security input, preprocessing the data so that useful information is identified and extracted, and representing the corresponding forensics evidence in a concise and compact way. Moreover, the tool will perform some basic correlation of the data sources used in order to provide a warning about nodes that have been validated to exhibit consistent malicious behavior, and thus correspond to infected machines.

Requirements and Contact details

Kind of Work:	30% theoretical , 70% practical
Requirements:	programming in python/perl and java, programming in c/c++ desired, experience with linux, basics of communication networks and protocols
Supervisor:	Ilias Raftopoulos, ETZ G97, +41 44 632 70 50
Co-supervisor	Xenofontas Dimitropoulos, ETZ G90, +41 44 632 70 04
Professor:	Prof. Dr. Bernhard Plattner

Appendix B

Nmap Top 50 most often used ports

This is a list of the 50 most often used TCP ports in the Internet. For the 25 most often used ports, the service they usually provide is given. The list is based on work form the author of Nmap, Gordon Lyon [40].

21 – FTP
22 – SSH
23 – Telnet
25 – SMTP
26
53 – DNS
80 – HTTP
81
110 – POP3
111 – SunRPC
113
135 – Microsoft End Point Mapper
139 – NetBIOS
143 – IMAP
179
199
443 – HTTPS
445
465
514
515
548
554
587
646
993 – IMAP over SSL
995 – PO3 over SSL
1025
1026
1027
1433
1720
1723 – Microsoft PPTP
2000
2001
3306 – MySQL
3389 – Microsoft RDP
5060
5666
5900 – VNC
6001
8000
8008
8080 – HTTP alternate
8443
8888
10000
32768
49152
49154

Appendix C

OpenVAS example output

An extract of 4 pages of a 37 pages long report from a vulnerability scan against a single host is given. The scanned host is a virtual machine which runs multiple vulnerable software to assess the performance and detectin rate of vulnerability scanners.

Summary

This document reports on the results of an automatic security scan. The report first summarises the results found. Then, for each host, the report describes every issue found. Please consider the advice given in each description, in order to rectify the issue.

Overrides are on. When a result has an override, this report uses the threat of the override.

Notes are included in the report.

This report might not show details of all issues that were found. It only lists hosts that produced issues. Issues with the threat level "Debug" are not shown.

This report contains all 109 results selected by the filtering described above. Before filtering there were results.

Scan started: Mon Mar 21 15:05:01 2011
Scan ended: Mon Mar 21 15:09:18 2011

Host Summary

Host	High	Medium	Low	Log	False Positive
172.16.68.128	25	34	40	10	0
Total: 1	25	34	40	10	0

Results per Host

Host 172.16.68.128

Scanning of this host started at: Mon Mar 21 15:05:06 2011
Number of results: 109

Port Summary for Host 172.16.68.128

Service (Port)	Threat Level
distcc (3632/tcp)	High
ftp (21/tcp)	High
http (80/tcp)	High
microsoft-ds (445/tcp)	High
mysql (3306/tcp)	High
domain (53/udp)	Medium
netbios-ns (137/udp)	Medium
ssh (22/tcp)	Medium
ajp13 (8009/tcp)	Low
domain (53/tcp)	Low
general/SMBClient	Low
general/tcp	Low
netbios-ssn (139/tcp)	Low
postgresql (5432/tcp)	Low

smtp (25/tcp)	Low
telnet (23/tcp)	Low
general/CPE-T	Log
general/HOST-T	Log

Security Issues for Host 172.16.68.128

High distcc (3632/tcp)
NVT: DistCC Detection (OID: 1.3.6.1.4.1.25623.1.0.12638)

```
distcc is a program to distribute builds of C, C++, Objective C or
Objective C++ code across several machines on a network.
distcc should always generate the same results as a local build, is simple
to install and use, and is often two or more times faster than a local compile.
distcc by default trusts its clients completely that in turn could
allow a malicious client to execute arbitrary commands on the server.
For more information about DistCC's security see:
http://distcc.samba.org/security.html
Risk factor : High
```

High (CVSS: 6.8) ftp (21/tcp)
NVT: ProFTPD Server SQL Injection Vulnerability (OID: 1.3.6.1.4.1.25623.1.0.900507)

```
  Overview: This host is running ProFTPD Server and is prone to remote
  SQL Injection vulnerability.
  Vulnerability Insight:
  This flaw occurs because the server performs improper input sanitising,
  - when a %(percent) character is passed in the username, a single quote
    (') gets introduced during variable substitution by mod_sql and this
    eventually allows for an SQL injection during login.
  - when NLS support is enabled, a flaw in variable substition feature in
    mod_sql_mysql and mod_sql_postgres may allow an attacker to bypass
    SQL injection protection mechanisms via invalid, encoded multibyte
    characters.
  Impact:
  Successful exploitation will allow remote attackers to execute arbitrary
  SQL commands, thus gaining access to random user accounts.
  Affected Software/OS:
  ProFTPD Server version 1.3.1 through 1.3.2rc2
  Fix:
  Upgrade to the latest version 1.3.2rc3,
  http://www.proftpd.org/
  References:
  http://www.milw0rm.com/exploits/8037
  http://www.securityfocus.com/archive/1/archive/1/500833/100/0/threaded
  http://www.securityfocus.com/archive/1/archive/1/500851/100/0/threaded
  CVSS Score:
    CVSS Base Score     : 6.8 (AV:N/AC:M/Au:NR/C:P/I:P/A:P)
    CVSS Temporal Score : 5.3
  Risk factor: High
CVE : CVE-2009-0542, CVE-2009-0543
BID : 33722
```

High (CVSS: 7.5) http (80/tcp)

High mysql (3306/tcp)
NVT: MySQL 5.x Unspecified Buffer Overflow Vulnerability (OID: 1.3.6.1.4.1.25623.1.0.100271)

Overview:
MySQL is prone to a buffer-overflow vulnerability because if fails to
perform adequate boundary checks on user-supplied data.
An attacker can leverage this issue to execute arbitrary code within
the context of the vulnerable application. Failed exploit attempts
will result in a denial-of-service condition.
This issue affects MySQL 5.x; other versions may also be vulnerable.
References:
http://www.securityfocus.com/bid/36242
http://www.mysql.com/
http://intevydis.com/company.shtml
Risk factor : High
BID : 36242

Medium (CVSS: 7.6) domain (53/udp)
NVT: ISC BIND 9 DNSSEC Bogus NXDOMAIN Response Remote Cache Poisoning Vulnerability (OID: 1.3.6.1.4.1.25623.1.0.100458)

Overview:
ISC BIND 9 is prone to a remote cache-poisoning vulnerability.
An attacker may leverage this issue to manipulate cache data,
potentially facilitating man-in-the-middle, site-impersonation, or denial-of-
service attacks.
Versions prior to the following are vulnerable:
BIND 9.4.3-P5 BIND 9.5.2-P2 BIND 9.6.1-P3
Solution:
Updates are available. Please see the references for details.
References:
http://www.securityfocus.com/bid/37865
http://www.isc.org/products/BIND/
http://www.kb.cert.org/vuls/id/360341
https://www.isc.org/advisories/CVE-2010-0097
Risk factor : Medium
CVE : CVE-2010-0097, CVE-2010-0290, CVE-2010-0382
BID : 37865

Medium (CVSS: 5.0) domain (53/udp)
NVT: OpenSSL DSA_verify() Security Bypass Vulnerability in BIND (OID: 1.3.6.1.4.1.25623.1.0.800338)

Overview: The host is running BIND and is prone to Security Bypass
Vulnerability.
Vulnerability Insight:
The flaw is caused due to improper validation of return value from OpenSSL's
DSA_do_verify and VP_VerifyFinal functions.
Impact:
Successful exploitation could allow remote attackers to bypass the certificate
validation checks and can cause man-in-the-middle attack via signature checks
on DSA and ECDSA keys used with SSL/TLS.
Impact Level: Application
Affected Software/OS:
ISC BIND version prior to 9.2 or 9.6.0 P1 or 9.5.1 P1 or 9.4.3 P1 or 9.3.6 P1/Linux

CVE : CAN-1999-0621

Medium (CVSS: 2.6) ssh (22/tcp)
NVT: OpenSSH CBC Mode Information Disclosure Vulnerability (OID: 1.3.6.1.4.1.25623.1.0.100153)

Overview: The host is installed with OpenSSH and is prone to information
disclosure vulnerability.
Vulnerability Insight:
The flaw is caused due to the improper handling of errors within an SSH session
encrypted with a block cipher algorithm in the Cipher-Block Chaining 'CBC' mode.
Impact:
Successful exploits will allow attackers to obtain four bytes of plaintext from
an encrypted session.

Impact Level: Application
Affected Software/OS:
Versions prior to OpenSSH 5.2 are vulnerable. Various versions of SSH Tectia
are also affected.
Fix: Upgrade to higher version
http://www.openssh.com/portable.html
References:
http://www.securityfocus.com/bid/32319
Risk factor: Medium
CVE : CVE-2008-5161
BID : 32319

Low ajp13 (8009/tcp)
NVT: Identify unknown services with nmap (OID: 1.3.6.1.4.1.25623.1.0.66286)

nmap thinks ajp13 is running on this port

Low distcc (3632/tcp)
NVT: Identify unknown services with nmap (OID: 1.3.6.1.4.1.25623.1.0.66286)

nmap thinks distccd is running on this port

Low domain (53/tcp)
NVT: DNS Server Detection (OID: 1.3.6.1.4.1.25623.1.0.100069)

Overview:
A DNS Server is running at this Host.
A Name Server translates domain names into IP addresses. This makes it
possible for a user to access a website by typing in the domain name instead of
the website's actual IP address.
Risk factor : None

Low domain (53/tcp)
NVT: Determine which version of BIND name daemon is running (OID: 1.3.6.1.4.1.25623.1.0.10028)

BIND 'NAMED' is an open-source DNS server from ISC.org.
Many proprietary DNS servers are based on BIND source code.
The BIND based NAMED servers (or DNS servers) allow remote users
to query for version and type information. The query of the CHAOS

Appendix D

Features used for automated Classification

In the following table, all features extracted during the analysis of security incidents from 7 days in June 2011 are given. In the first column, the name of the feature is printed. The second column holds the source of information from which this feature was extracted. In the third column the type (binary, nominal or real) is given. The forth and fifth column show the number of cases in which the feature was found and, if applicable, the average value of the feature.

Feature Name	Source	Type	Occurrences	Average value
blacklist-bot	blacklist	Real	11	3.3
blacklist-malware	blacklist	Real	19	3.7
blacklist-multiple	blacklist	Real	121	6.3
blacklist-nohits	blacklist	Real	5	1.0
blacklist-rbn	blacklist	Real	85	5.7
dns-tag	hostname	Nominal	65	0.0
dst-port-22	IDS	Binary	1	
dst-port-25	IDS	Binary	6	
dst-port-443	IDS	Binary	9	
dst-port-53	IDS	Binary	2	
dst-port-80	IDS	Binary	81	
dst-port-82	IDS	Binary	1	
dst-port-HIGH	IDS	Binary	78	
google-blacklist	Search Engine	Real	20	6.5
google-bot	Search Engine	Real	64	8.0
google-malware	Search Engine	Real	55	10.6
google-p2p	Search Engine	Real	23	9.9
google-server	Search Engine	Real	29	2.6
infection-dur	IDS	Real	80	201.0
service-OSLinux	Network Scan	Binary	8	
service-OSWindows	Network Scan	Binary	15	
service-OSiOS	Network Scan	Binary	1	
service-X11:1	Network Scan	Binary	1	
service-domain	Network Scan	Binary	1	
service-dynamic_ip_wont_scan	Network Scan	Binary	61	
service-ftp	Network Scan	Binary	2	
service-hostdown	Network Scan	Binary	26	
service-http	Network Scan	Binary	17	
service-http-alt	Network Scan	Binary	1	
service-http-proxy	Network Scan	Binary	1	
service-https	Network Scan	Binary	4	
service-imap	Network Scan	Binary	2	
service-imaps	Network Scan	Binary	1	
service-microsoft-ds	Network Scan	Binary	1	
service-microsoft-rdp	Network Scan	Binary	5	
service-ms-term-serv	Network Scan	Binary	2	
service-msrpc	Network Scan	Binary	3	
service-mysql	Network Scan	Binary	2	
service-netbios-ssn	Network Scan	Binary	5	
service-no_open_ports	Network Scan	Binary	5	
service-pop3	Network Scan	Binary	1	
service-pop3s	Network Scan	Binary	1	
service-rpcbind	Network Scan	Binary	7	
service-rsync	Network Scan	Binary	1	
service-shell	Network Scan	Binary	2	
service-skype2	Network Scan	Binary	1	
service-smtp	Network Scan	Binary	6	
service-smtps	Network Scan	Binary	1	
service-snet-sensor-mgmt	Network Scan	Binary	1	
service-ssh	Network Scan	Binary	12	
service-submission	Network Scan	Binary	2	
service-tcpwrapped	Network Scan	Binary	2	
service-vnc	Network Scan	Binary	2	
snort-attack	IDS	Real	23	537.0
snort-compromised	IDS	Real	44	10180.9
snort-policy	IDS	Real	80	8981.0
snort-scan	IDS	Real	0	

Feature Name	Source	Type	Occurrences	Average value
src-port-1024	IDS	Binary	1	
src-port-21	IDS	Binary	1	
src-port-25	IDS	Binary	1	
src-port-26	IDS	Binary	1	
src-port-445	IDS	Binary	1	
src-port-53	IDS	Binary	4	
src-port-80	IDS	Binary	9	
src-port-HIGH	IDS	Binary	115	
topsnort-2000345	IDS	Binary	5	
topsnort-2000347	IDS	Binary	2	
topsnort-2000348	IDS	Binary	2	
topsnort-2002167	IDS	Binary	1	
topsnort-2002728	IDS	Binary	1	
topsnort-2003219	IDS	Binary	4	
topsnort-2003620	IDS	Binary	1	
topsnort-2005320	IDS	Binary	2	
topsnort-2006384	IDS	Binary	1	
topsnort-2007626	IDS	Binary	5	
topsnort-2007668	IDS	Binary	1	
topsnort-2008321	IDS	Binary	1	
topsnort-2008374	IDS	Binary	1	
topsnort-2008411	IDS	Binary	2	
topsnort-2008428	IDS	Binary	1	
topsnort-2009024	IDS	Binary	4	
topsnort-2009867	IDS	Binary	2	
topsnort-2010071	IDS	Binary	1	
topsnort-2011365	IDS	Binary	1	
topsnort-2011849	IDS	Binary	1	
topsnort-2011912	IDS	Binary	1	
topsnort-2012113	IDS	Binary	1	
topsnort-2012401	IDS	Binary	1	
topsnort-2012609	IDS	Binary	1	
topsnort-2012612	IDS	Binary	1	
topsnort-2012625	IDS	Binary	1	
topsnort-2012627	IDS	Binary	1	
topsnort-2012645	IDS	Binary	2	
topsnort-2012686	IDS	Binary	2	
topsnort-2012725	IDS	Binary	1	
topsnort-2012863	IDS	Binary	1	
topsnort-2012939	IDS	Binary	3	
topsnort-2012958	IDS	Binary	1	
topsnort-2012959	IDS	Binary	1	
topsnort-2013121	IDS	Binary	1	
topsnort-2400001	IDS	Binary	53	
topsnort-2801246	IDS	Binary	3	
topsnort-2801953	IDS	Binary	3	
topsnort-2801954	IDS	Binary	3	
topsnort-2802841	IDS	Binary	1	
topsnort-2802870	IDS	Binary	1	
topsnort-2802895	IDS	Binary	2	
topsnort-2802912	IDS	Binary	1	
topsnort-2802929	IDS	Binary	1	
topsnort-2802975	IDS	Binary	15	
topsnort-2803032	IDS	Binary	1	
topsnort-369	IDS	Binary	1	
topsnort-count	IDS	Real	93	173.8

Appendix E

Keywords used to generate Tags

The full list of keywords which were used in automated analysis of search results and hostnames.

List of words used for classifying everything except hostnames:

Tag	Keywords
Server	ftp,webmail,email,proxy,smtp,mysql,pop3,mms,netbios,irc
DynamicUser	dhcp
Malicious	malware,spybot
Spam	spam
Blacklisted	blacklist,ban,banlist,blocklist
Adware	adware
Bot	bot,trojan,worm
ircServer	irc,undernet,innernet
P2P	torrent,emule,kazaa,edonkey,announce,tracker
	xunlei,limewire,bitcomet,uusee,qqlive,pplive

List of words used for classifying hostnames:

Tag	Keywords
Server	ftp,webmail,email,proxy,smtp,mysql,pop3,mms
	netbios,irc,server,serv,srv
DynamicUser	dhcp,public
Malicious	malware,spybot
dnsServer	dns
natGateway	nat

Bibliography

[1] Dmitri Alperovitch. Revealed: Operation shady rat. http://www.mcafee.com/us/resources/white-papers/wp-operation-shady-rat.pdf.

[2] Jürgen Kuri Jürgen Schmidt Jan-Keno Janssen. Operation payback: protests via mouse click. http://www.h-online.com/security/news/item/Operation-Payback-protests-via-mouse-click-1150790.html. Last visited: 2011-08-24.

[3] The H. Lulzsec comes back to hack the sun web site. http://www.h-online.com/security/news/item/Operation-Payback-protests-via-mouse-click-1150790.html. Last visited: 2011-08-24.

[4] The H. Anonymous dump 7.4 gb of us law enforcement web sites. http://www.h-online.com/security/news/item/Anonymous-dump-7-4-GB-of-US-law-enforcement-web-sites-1320117.html. Last visited: 2011-08-24.

[5] The H. Hacktivists break into sony pictures database. http://www.h-online.com/security/news/item/Hacktivists-break-into-Sony-Pictures-database-1254622.html. Last visited: 2011-08-24.

[6] Omar Santos. *End-to-End Network Security: Defense-in-Depth*. Cisco Press, 2007.

[7] Chris McNab. *Network Security Assessment: Know Your Network*. O'Reilly Media, 2007.

[8] Alienvault. http://alienvault.com/. Last visited: 2011-08-24.

[9] OSSIEM. http://alienvault.com/community. Last visited: 2011-08-24.

[10] SNORT. http://www.snort.org/. Last visited: 2011-08-24.

[11] Cécile Lüssi. Signature-based extrusion detection. Master's thesis, ETH Zurich, 2008.

[12] Elias Raftopoulos and Xenofontas Dimitropoulos. Detecting, validating and characterizing computer infections from IDS alerts. TIK-Report 337, ETH Zurich, June 2011.

[13] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, et al. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, pages 12–26. IEEE, 2000.

[14] J. McHUGH. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[15] Mcafee mistakenly detects legitimate windows system files as malicious in false positive nightmare. http://www.scmagazineuk.com/mcafee-mistakenly-detects-legitimate\-windows-system-files-as-malicious-in-false-positive-nightmare/article/168521/. Last visited: 2011-08-26.

[16] ThreatExpert. Threatexpert is an advanced automated threat analysis system... http://www.threatexpert.com/. Last visited: 2011-08-27.

[17] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (idps). *NIST Special Publication*, 800(2007):94, 2007.

[18] Sourcefire. http://www.sourcefire.com/. Last visited: 2011-08-26.

[19] Emergingthreats. http://www.emergingthreats.net/. Last visited: 2011-08-26.

[20] Bleedingsnort snort ruleset. http://www.bleedingsnort.com/. Last visited: 2011-09-03.

[21] Internet Assigned Numbers Authority (IANA). Port numbers. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml. Last visited: 2011-08-27.

[22] nmap nmap. Network scanner. http://www.nmap.org.

[23] B. Claise and S. Bryant. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information. Technical report, RFC 5101, January, 2008.

[24] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast ip networks. 2005.

[25] Brian Trammell, Elisa Boschi, Gregorio Procissi, Christian Callegari, Peter Dorfinger, and Dominik Schatzmann. Identifying skype traffic in a large-scale flow data repository. In *4th Traffic Measurement and Analysis Workshop*, Vienna, Austria, 2011.

[26] GFI. Gfi sandbox. http://www.gfi.com/malware-analysis-tool/. Last visited: 2011-08-28.

[27] COMODO. Instant malware analysis. http://camas.comodo.com/. Last visited: 2011-08-28.

[28] Robtex. Swiss army knife internet tool. http://www.robtex.com/.

[29] Common Vulnerabilities and Exposures. http://cve.mitre.org/. Last visited: 2011-08-28.

[30] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovi, and Antonio Nucci. Unconstrained endpoint profiling (googling the internet). *SIGCOMM Comput. Commun. Rev.*, 38:279–290, August 2008.

[31] Zeus Tracker. https://zeustracker.abuse.ch/. Last visited: 2011-08-26.

[32] SpyEye Tracker. https://spyeyetracker.abuse.ch/. Last visited: 2011-08-28.

[33] ShadowServer. http://www.shadowserver.org/. Last visited: 2011-08-28.

[34] Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 342–351, New York, NY, USA, 2007. ACM.

[35] ETH Zürich. Jahresbericht 2010. http://www.ethz.ch/about/publications/annualreports/eth_jahresbericht_2010_en.pdf. Last visited: 2011-09-05.

[36] Informatikdienste. Statistiken 2010. https://www1.ethz.ch/id/about/facts. Last visited: 2011-09-05.

[37] OpenVAS. http://www.openvas.org/. Last visited: 2011-08-26.

[38] Tenable Network Security. Nessus vulnerability scanner. http://www.tenable.com/products/nessus. Last visited: 2011-08-30.

[39] Sqlite db performance. http://diegopizzocaro.posterous.com/sqlite-search-performances. Last visited: 2011-09-03.

[40] G.F. Lyon. Nmap: Scanning the internet. https://www.blackhat.com/presentations/bh-usa-08/Vaskovich/BH_US_08_Vaskovich_Nmap_Scanning_the_Internet.pdf.

[41] Y. Fyodor. Remote os detection via tcp/ip stack fingerprinting, 1998.

[42] Google. Custom search api. http://code.google.com/apis/customsearch/v1/overview.html. Last visited: 2011-09-05.

[43] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, page 28, 1998.

[44] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2):131–163, 1997.

[45] J.R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[46] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[47] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[48] University of Waikato. Attribute-relation file format (arff). http://www.cs.waikato.ac.nz/ml/weka/arff.html. Last visited: 2011-08-30.

[49] Flask Python Microframework. http://flask.pocoo.org/. Last visited: 2011-08-30.

[50] CERT Network Situational Awareness Team. System for internet-level knowledge silk. http://tools.netsa.cert.org/silk/index.html. Last visited 2011-08-30.

[51] Python nmap module. http://xael.org/norman/python/python-nmap/. Last visited: 2011-08-30.

[52] Jinja2 Template Engine. http://jinja.pocoo.org/. Last visited: 2011-08-30.

[53] Ask Search Toolbar. http://toolbar.ask.com/. Last visisted: 2011-08-30.

[54] LibSVM. http://www.csie.ntu.edu.tw/~cjlin/libsvm/. Last visited: 2011-08-30.

[55] J.C. Platt. Using analytic qp and sparseness to speed training of support vector machines. *Advances in Neural Information Processing Systems*, pages 557–563, 1999.