NTNU
Det skapende universitet

# Bruk av kunstig intelligens for å oppdage innbrudd i datasystemer

Ole Morten Grodås

Master i datateknikk
Innlevert: Juni 2012
Hovedveileder: Tor Stålhane, IDI
Medveileder: Torgeir Broen, Forsvarets forskningsinstitutt

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

## Problem description

With its large growth, the Internet has emerged as an area for organized crime. An important aspect of cybercrime is breaking into computer systems and securing control over them. Many of today's most widely used network intrusion detection systems are based on hand-crafted signatures. With the increasing volume of malicious activity, it is getting harder and harder to keep manually written signature sets up to date. The problem has been further exacerbated by the fact that many cybercriminals work hard to avoid detection.

The research literature describes a number of systems to automate intrusion detection based on artificial intelligence, but few of these systems are in operational use today. A problem often cited as the reason why these systems are not used is a high false positive rate.

This thesis aims to design and implement a prototype intrusion detection system based on artificial intelligence. The system should as far as possible automate the process of detecting compromised computers. The system will be evaluated on real network traffic in order to assess the effectiveness in the best possible way.

Supervised by:

Tor Stålhane, Professor at Norwegian University of Science and Technology (NTNU)

Torgeir Broen, Scientist at Norwegian Defence Research Establishment (FFI)

## Abstract

With its large growth, the Internet has emerged as an area for organized crime. As other types of organized crime most of the activity is motivated by economic profits. In addition to economically motivated threat agents some are driven by political motives, like national state intelligence agencies and cyber terrorist. An important aspect of cybercrime is breaking into computer systems and securing control over them. When cyber criminals have successfully gained access to a system they often install a hidden malicious program to secure permanent access. The program is normally a bot that recruits the compromised computer into a botnet. Multiple bots under a common central administration is called a botnet.

This paper describes the design of a botnet detector and reports the results from testing the detector on real world data from an organization in Norway. The proposed system is designed around automation of a classical misuse detection system. It takes as input activity log from the network like netflow, DNS log or HTTP log and searches this log data using a large set of signatures. The signature set is generated by algorithmically combining freely available signature lists. Most of the signatures are either automatically generated by an external mechanism, for example by running malware samples in a sandbox or created and shared by information security communities. The detector is based on four main components. 1) An algorithm for quantifying the risk represented by a signature, 2) An algorithm for whitelisting bad signatures that would create false positives, 3) A matching engine for searching log files with a large signature set, and 4) An algorithm for identifying compromised computers by aggregating alarm data.

Putting all the components together the system provides a significant improvement over intrusion detection based on searching logs using blacklist with simple string search algorithms. The two main improvements are 1) The system makes it easier to deal with large signature sets because most bad signatures are automatically whitelisted; 2) Alarms have a quantified risk score and are aggregated up to a victim risk score. This reduces the need to manually inspect each alarm and simplifies the job of identifying compromised computers.

The system is complementary and synergistic to some of the recently suggested system in the research literature like Exposure(Bilge, Kirda, Kruegel, & Balduzzi, 2011) and Notos(Antonakakis, Perdisci, Dagon, Lee, & Feamster, 2010)

## Abstract (Norwegian)

Med sin store vekst, har internett utviklet seg til et lukrativt domene for organisert kriminalitet. Som andre typer organisert kriminalitet er mesteparten av aktiviteten motivert av økonomisk gevinst. I tillegg til økonomisk motiverte trusseleaktører er noen tilsynelatende drevet av politiske motiver, som nasjonalstaters etterretningsorganisasjoner og cyberterrorister. En viktig del av datakriminalitet er å bryte seg inn i datasystemer og sikre fremtidig kontroll over systemene. Når nettkriminelle har klart å få tilgang til et system installerer de ofte et skjult program for å sikre fremtidige tilgang. Dette programmet kalles en bot og rekrutterer den kompromitterte maskinen inn i et botnet. Flere boter under en felles sentral administrasjon kalles et botnet.

Denne oppgaven beskriver utformingen av en botnet detektor og rapporterer resultatene fra testing av detektoren på reelle data fra en organisasjon i Norge. Det foreslåtte systemet er designet rundt automatisering av et klassisk "misuse detection system". Det tar som input nettverksaktivitetslogg som for eksempel NetFlow, DNS logg og HTTP logg og søker igjennom denne loggen med et stort signatursett. Signatursettet er generert ved å algoritmisk kombinere fritt tilgjengelige signaturlister. De fleste signaturlistene er enten automatisk generert av en ekstern mekanisme, for eksempel ved å kjøre malware i en sandbox eller laget og delt av ulike informasjonssikkerhet miljøer. Detektoren er basert på fire hovedkomponenter. 1) En algoritme for å kvantifisere risikoen representert ved en signatur, 2) En algoritme for fjerning av dårlige signaturer som vil skapt mange falske positive, 3) En søkemotor for å søke igjennom loggfiler med et stort signatursett, og 4) En algoritme for å identifisere kompromittert datamaskiner ved å aggregere alarm data.

Ved å se alle komponentene under ett ser det ut til at systemet gir en betydelig forbedring i forhold til inntrengningsdeteksjon basert på vanlig signatursøk. De to viktigste forbedringene er at 1) Systemet gjør det lettere å håndtere store signatur sett fordi de fleste feilaktige signaturene blir automatisk fjernet og 2) Alarmer har en kvantifisert risiko og aggregeres opp til en risiko for hver klient i nettverket. Dette reduserer behovet for og manuelt inspisere hver enkelt alarm og forenkler dermed jobben med å identifisere kompromitterte datamaskiner

Systemet er komplementært og synergistisk med noen av de nylig foreslåtte systemene i forskningslitteraturen som Exposure(Bilge et al., 2011) og Notos (Antonakakis et al., 2010)

## Acknowledgements

First, I would like to thank my professor, Tor Stålhane for encouragement and providing good advice and feedback.

I would also like to thank my supervisor Torgeir Broen, Scientist at the Norwegian Defence Research Establishment, for providing good advice and feedback on the work as well as advice on writing a great thesis.

I would also like to point out the great support I have received from my current employer Norsk Helsenett SF and I especially want to thank my colleagues at the Norwegian Healthcare Computer Security Incident Response Team (HelseCSIRT): Jørgen Bøhnsdalen, Kjell Christian Nilsen, Kjell Tore Fossbakk and Suhail Mushtaq for interesting discussions that provided important insights and for feedback on the report.

Trondheim, June 2012

Ole Morten Grodås

# Contents

# 1 Introduction

This paper describes the design of a botnet detector and reports the results from testing the detector on real world data from an organization in Norway.

## 1.1 Motivation

With its large growth, the Internet has emerged as an area for organized crime. The problems of cybercrime has grown to such a magnitude that the United Nations treats it at the same level as other major organized crime activities like smuggling and trafficking. Estimated annual loss from cybercrime range from 1 billion USD to 1 trillion USD (*Globalization of Crime: A Transnational Organized Crime Threat Assessment*, 2010).

Cybercrime is a broad term and includes many activities and many actors. The original hacker stereotype as a smart teenage or adult male that breaks into computers for fun and fame is no longer a good description of today's cybercriminals. As other types of organized crime most of the activity is motivated by economic profits. In addition to economically motivated threat agents some might be driven by political motives, for example national state intelligence agencies and cyber terrorist.

An important aspect of cybercrime is breaking into computer systems and securing control over them. When cyber criminals have successfully gained access to a system they often install a hidden malicious program to ensure permanent access. The program is normally a bot that recruits the compromised computer into a botnet.

Botnets are an underlying tool used by most cybercriminals from economically motivated criminals to national states conducting cyber warfare. Examples of two famous botnets are Stuxnet and Spyeye. Stuxnet became famous because of the speculation that it was designed by a nation state to target the Iraninan Bushehr and Natanz nuclear power plant. Spyeye is a powerful botnet kit which has been used, among other things, to target multiple norwegian banks.

The UN Transnational Organized Crime Threat Assessment concludes that cyber criminals are opportunistic and predatory. Law enforcement strategies based on arrests and seizures alone are not likely to be successful because new criminals will pop up as soon as the old ones are removed. They conclude that *“the solutions are more likely to be technological, aimed at making it more difficult to acquire money in these ways.”,* finding better techniques for detecting and tracking botnets will probably be helpful in this regard. If the malicious activity is detected in an early stage, the victims can be notified before any harm is done. Better detection and tracking also has the potential to assist law enforcement in tracking down and arresting the criminals.

## 1.2 Botnet history and terminology

The name botnet has a rather long history. The first part of the word "bot" is a shorted form of “robots” and was originally a neutral, non-malicious term. The word seems to have been first used in 1920 by the Czech science fiction writer Karel Čapek(“Robot,” 2012). The term is a derivation from

the Czech word robota, which can be translated to English as “forced labour”, or more simply just “work”. In his writing, Čapek used the word robot for artificial people created in a factory. Even today, there does not seem to be a single agreed upon definition of the term robot. A broad characterization would be “A robot is a mechanical or virtual intelligent agent that can perform tasks automatically or with guidance, typically by remote control”. The term “robot” is usually used when referring to an electro-mechanical machine, while the term “bot” is used when referring to a software agent. The term bot, as in software agent, was first used to describe a virtual individual that could sit on an Internet Relay Chat (IRC) channel and do things for its owner while the owner was busy elsewhere. The first IRC bot was developed in 1988 by Jarkko Oikarinen of the University of Olulu (Amit, 2011). As the usage of IRC grew, enthusiasts began adding more features to the bots, for example, automated scripts for logging channel statistics, providing mechanisms for file distribution and exercising operator privileges.

The term *botnet* was also initially a non-malicious term and has it origin from the early days of IRC networks. On most IRC networks, each channel has one or more channel operators, which has extra privileges. If the channel operator is taken offline, another member of the channel will automatically be assigned operator status. In the early 1990s, malicious users began exploiting this behavior in attempts to acquire operator status; this kind of attack is often called channel takeover attacks. The most popular technique at the time was to perform a denial of service (DOS) attack against the channel operator by flooding his uplink. In December 1993, Robey Pointer developed the IRC bot eggdrop, which was designed to help manage and protect channels from takeover attempts and other forms of IRC war. It was written in C, but designed to allow execution of user added TCL scripts. Eggdrop had a key feature called *botnet* that enabled multiple eggdrop bots to exchange information. This allowed eggdrop bots to share user lists, ban lists, ignore lists and effectively protect channels against takeover attempts. It’s doubtful that the author ever envisaged its architecture being put to malicious use, controlling networks of tens of thousands zombies (Canavan, 2005). Today the term botnet usually refers to a collection of compromised computers connected to the internet. Earlier botnets typically used the IRC protocol, but today's botnets use a variety of protocols. Currently one of the most popular protocols is HTTP. One of the reasons HTTP is so popular is that most firewalls does not block outbound HTTP traffic.(Li, 2009)

The term bot can refer to both the compromised computer and the malicious software running on the compromised computer, depending on the context. It is also worth mentioning that in some situations the term bot are still used to refer to "good bots", examples are web crawlers, software update agents, and artificial characters in computer games. For the remainder of this paper the term bot will imply a “bad bot”.

Bad bots are part of a broader term for software called malicious software or malware for short. The term is used to describe most types of hostile, intrusive or annoying software. Malware includes computer viruses, Trojan horses, spyware, dishonest adware, rootkits, key loggers, botnets and more. There has been attempts to create taxonomies for malware, but experience has demonstrated that it is very difficult to create mutually exclusive or unambiguous categories of malware based on the commonly used terminology (Karresand, 2002).

The following is a short summary of the terminology used to describe botnets and related malware. A malware program with a unique MD5 sum is called a malware sample. The best way to interpret the traditional malware classification terminology like virus, worm, trojans etc is that that they describe features. A single malware sample can have multiple features at the same time. A typical malware sample can therefore be a virus, a worm, a rootkit and a bot at the same time. On recent example of this is the botnet known as TDL or Alureon , which also had advanced rootkit functionality[1]

Viruses and worms can collectively be described as infectious malware and are known for the manner in which they spread, rather than any other particular behavior. Typically, a virus attaches itself to executable files that may be part of legitimate programs. When the user launches the infected program, the virus code is also executed and the virus spreads itself to other files. A worm is also self-replicating program, but a worm typically spreads by sending copies of itself to other nodes in the network. Often a worm is spreading without user intervention due to security vulnerabilities in operating systems or other software. Unlike a virus, a worm does not need to attach itself to an existing program.

Non-infectious malware have to rely on other methods for infections. A popular technique is phishing. The term is a variation of the word fishing, and alludes to “baiting” users in hopes that the potential victim will “bite” by clicking a malicious link, opening a malicious attachment in which case the attacker gains access to the computer system or confidential information. A popular type of non-infectious malware is a Trojan horse, which can be defined as a program that invites the user to run it, concealing a harmful or malicious payload.

The code used to take advantage of vulnerabilities in computer systems is an exploit. After a computer is compromised, the attacker typically wants to ensure continued access. This is normally done by installing a bot on the computer. The software used to download and install the bot is called a dropper. The server the bot contacts for getting new instructions is a command and control server. If the bot is exfiltrating data, the server the data is exfiltrated too is often called a drop site. Developing bots and exploits is very resource intensive, for this reason many botnet operators does not make their own, but buy ready-made commercial botnet kits or exploit kits. Bots that are based on the same botnet kits are considered part of the same botnet family.

## 1.3 Botnet overview

The main purpose of a botnet is to allow the owner of the botnet to control computers he has compromised. In the early days before stateful firewalls were common, cyber-attacks were typically performed by directly attacking services running on the victim’s computer. To secure future access the attacker would open a port that gave him shell on the compromised client and use it as a backdoor into the system. This simple attack is illustrated in the figure 1 below.

[1] An analys of the Alureon rootkit functionality by Microsoft Software Development Engineer Joe Johnson http://www.virusbtn.com/pdf/conference_slides/2010/Johnson-VB2010.pdf

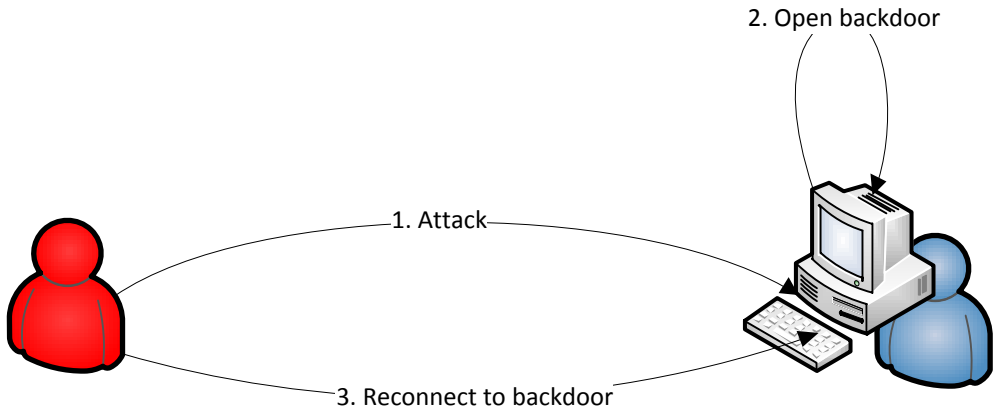



Figure 1: Cybercriminal directly attack the victims computer

These direct attacks were possible because many computers were running services with known vulnerabilities by default. Opening a backdoor and connecting to it was possible because stateful firewalls were not very common. With the increasing focus on security, most clients stopped having open ports by default and most computers were put behind stateful firewalls. These security measures made the simple direct attacks inefficient.

Because of these security measures, cyber criminals developed more advanced indirect methods for attacking and maintain control of their victims. This development is the basis for today's botnets. The figure below illustrates a typical modern attack. Cyber criminals attack the user indirectly by first compromising a legitimate web page. The attacker maintains access by installing a bot that poll a command and control server for new instructions. These more indirect attack methods are illustrated in figure 2 below.

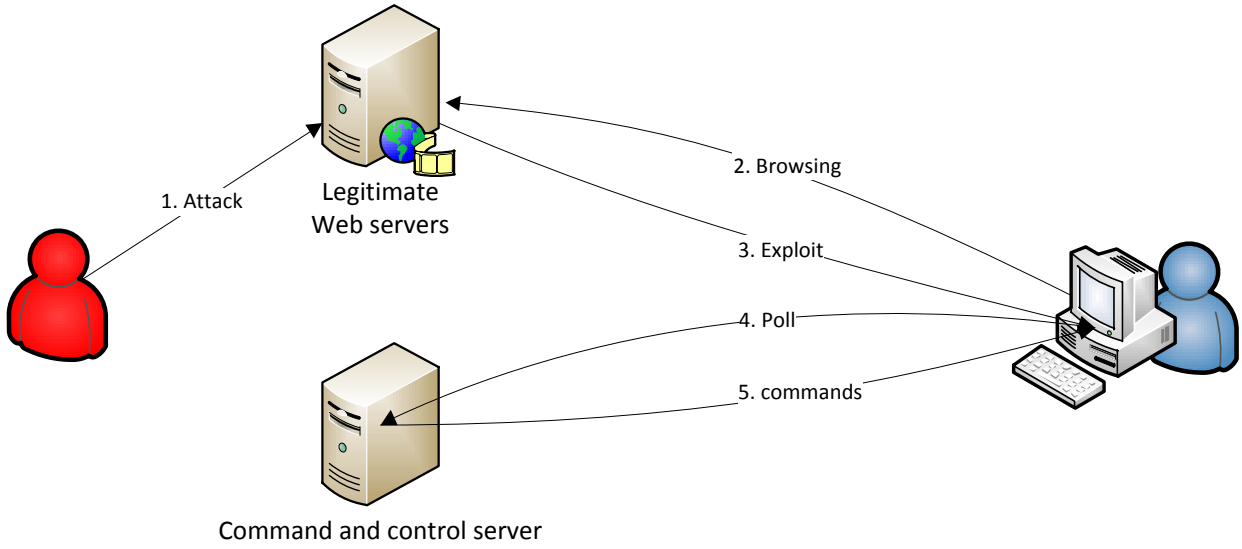



Figure 2 illustrates an example of how cybercriminals recruit computers into botnets

This attack method defeats the extra security that stateful firewalls provide. There are many variations of the attack describe above. Another very popular mechanism for recruiting clients into botnets besides hacking legitimate servers is sending phishing emails to the victim.

As botnets has been recognized as an increasing problem more efforts has been put into detecting and shutting them down. This again has led to efforts from criminals to create more robust botnets.(Ollmann, 2009)

## 1.4 Problem overview

Most network-based botnet tracking in operational use is either based on handcrafted full content signatures or blacklists of known malicious IP-addresses or domains. Both of these strategies have advantages and drawbacks.

The full content signatures are primarily tracking characteristics of the communication protocol used by the botnet software. Writing botnet software is a resource intensive activity and most botnets operators are therefore using standardized commercial botnet kits. Because of this, well-written full content signatures are able to detect many botnets as long as they are using the same botnet kit. This is the main advantage of full content signatures. The disadvantage is that developing high quality signatures requires substantial effort from a highly skilled security analyst. With the increasing volume of malicious activity, it is getting harder and harder to keep manually written signature sets up to date.

Writing good signatures are complicated by the fact that malware authors know that their activity is tracked using full content signatures. To mitigate this threat they have developed a series of techniques to defeat full content signatures. One of their techniques is to make the botnet traffic as similar to normal traffic as possible. This strategy makes it very difficult to write good signatures

with low false positive rates. Another strategy for malware authors is utilizing encrypted protocols like HTTPS. The use of encrypted protocols makes most full content signatures useless. Because it is easy for cyber-criminals to enable strong encryption, botnet detection techniques that rely on full content data will always be inherently unreliable.

Blacklists on the other hand are primarily tracking the network resources cyber-criminals are using. They can essentially be considered simplified signatures that only use the domain or IP-address of the Command and control server to identify botnets. Because the IP-address is needed by routers to forward traffic on the internet cyber-criminals cannot easily hide the IP-address with encryption in the same manner that the rest of the packet data can be hidden. The standard DNS protocol is also unencrypted making it difficult for criminals to hide the domain names they use as well.

The big advantage of blacklists is that the detection mechanism depends on the parts of the botnet traffic that is very difficult to hide for cyber-criminals. Another advantage is that they can more easily be updated automatically. For example by running malware samples in honeypots or by static analysis, the command and control server can be found and automatically added to blacklists. Keeping blacklists up to date does therefore not require the constant attention of a highly skilled security analyst in the same manner that full content signatures sets typically do.

IP and domain blacklist has therefore emerged as a very powerful tool for tracking botnets. Because of this, malware authors have come up with a series of strategies to defeat blacklist detection. One of the techniques is increased command and control server agility. In this context, agility means moving the botnet command and control server often, typically multiple times per day. Another strategy they are using to defeat blacklists is compromising legitimate servers and utilize them as command and control serves. This creates a problem for blacklisted based techniques because the server the cyber-criminals are using is both hosting benign and malicious content at the same time. Adding the server to the blacklist will therefore create false positives, but not adding the server to the blacklist will create false negatives. This illustrates that simple blacklists in some cases does not have enough expressiveness to distinguish malicious traffic from benign traffic.

In addition to these techniques, there are considerable efforts in the research community on finding good detecting methods based on machine learning and anomaly detecting methods. The limited use of machine learning and anomaly detecting in operational botnet detecting is there for somewhat surprising. The paper *"Outside the Closed World: On Using Machine Learning For Network Intrusion Detection"* (Sommer & Paxson, 2010a) argues that the field of network intrusion detecting exhibits inherit characteristics that makes it difficult to apply machine learning. The characteristics identified by Sommer et al are 1) The need for outlier detection, 2) High cost of classification errors, 3) A sematic gap between detection results and their operational meaning, 4) The enormous variability of benign traffic, 5) Challenges with performing sound evaluation, and 6) The need to operate in an adversarial setting.

## 1.5 System overview

The proposed system is inspired by the difficulty in applying machine learning to intrusion detecting outlined in *"Outside the Closed World: On Using Machine Learning For Network Intrusion Detection"* (Sommer & Paxson, 2010a). The proposed system takes as a starting point the manual methods for detecting botnets in operational use today and tries to automate the processes in such a way that compromised computers can be identified automatically. As part of this master thesis, a prototype system is implemented and its performance is tested on real word data. The proposed system is designed around a classical misuse detection system. Figure 3 below illustrates the basic concept.

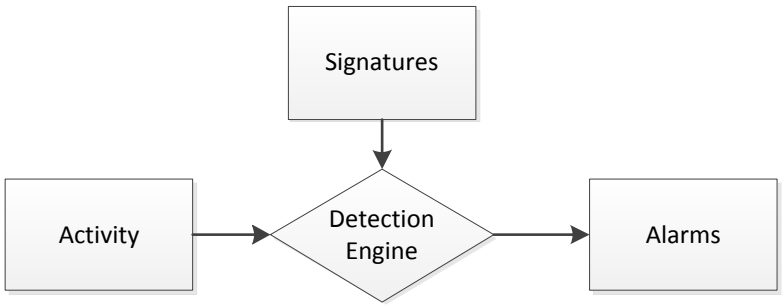



**Figure 3: Detection system overview**

The *Activity* can be any form of line-based log, but the most useful activity log in this context is netflow, DNS log and HTTP log. In a large organization, there are typically products from many vendors and each vendor has its own log format. In such scenarios, it simplifies the log management if an IDS sensor is used for passively generating log data based on the network data. When this is done, multiple sensor can be deployed for collecting logs and all logs will be in the same format. Typically, these sensors are placed between the corporate network and the Internet as described on in figure 4 below.

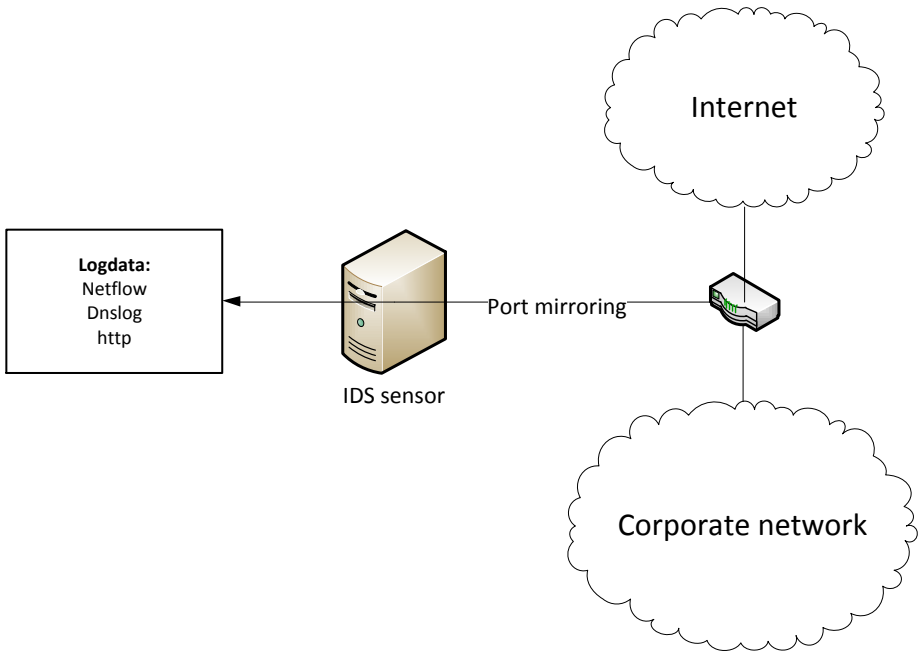



Figure 4: Illustrates activity log collection

In addition to the detection engine itself, the system has two important components: one for generating the signature set and one for managing the alarms from the detection engine.

The component that generates the signature is threat centric. It collects signatures that can detect malicious activity and estimates a risk score for each signature. At the most basic level the threat centric component takes as input multiple signature list and output a new combined signatures list were each signature is assigned a risk score. The quality of signature list and signature varies a lot and the challenge is to find a good method for quantifying a signature score. The figure below illustrates the basic workings of the threat centric component:

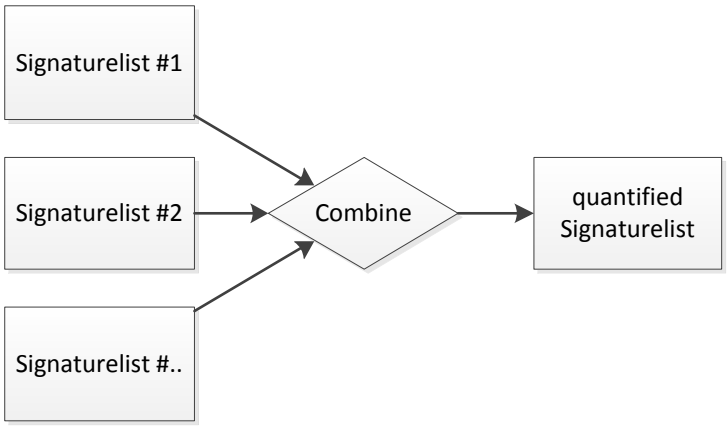



Figure 5: Process to dynamically learn network resources used by threat agents

The alarm management component is asset centric, it analyses the alarms generated for each client on the network and based on this estimates the likelihood that the client has been compromised. The figure below illustrates the basic workings of the asset centric component:

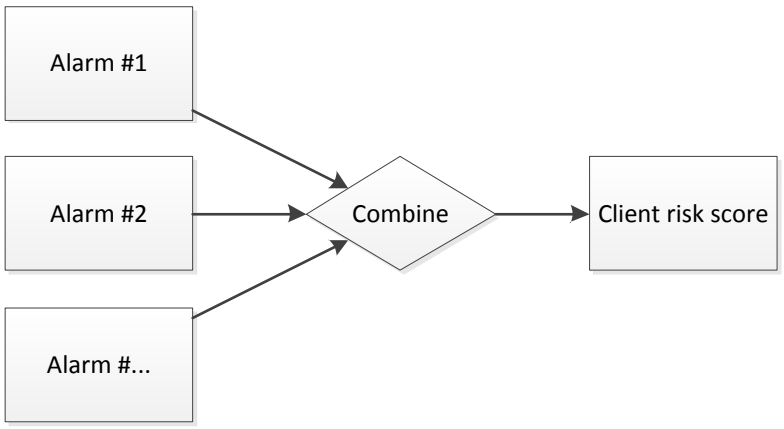



Figure 6: Process to dynamically assign a risk score to hosts in the network

These kinds of intrusion detection system can often create enormous amounts of alarms with a very high false positive rate. The important part of this module is to relieve some of the alarm management from the operator and instead present to the operator the list of clients that most likely are compromised.

## 1.6 Research question

Traditionally a security analyst has manually performed the tasks of downloading signatures, searching logs and analyzing the resulting alarms. The proposed system automates this process; the goal of the automation is to simplify the detection of compromised hosts in the network

One of the main challenges of using large signature sets is that they tend to generate alarms with a high false positive rate and this makes it easy to miss important alarms. If a signature triggers on a popular legitimate service it can even generate so many alarms that it floods the alarms management system. It is therefore very important that the proposed systems is able to deal with such bad bad signatures.

*RQ1: Will the system be able to automatically deal with signatures that generate large amount of false positives?*

The hypothesis is two-fold: 1) that a few bad signatures triggers on normal legitimate traffic and this generates most of the false positives and 2) that the proposed whitelisting algorithm in combination with the proposed matching engine will be able to automatically remove these signatures, there by significantly reducing the number of false positives.

To test the hypothesis the proposed system will be tested on network activity logs from a large organization in Norway. The log data will first be searched using the full signature set without the proposed whitelisting algorithm. The results will then be compared to searching the same log data after applying the whitelisting algorithm. The proposed matching engine will be tested the same way. First log data will be searched using a simple string matching algorithm and then the results will be compared to results of applying the proposed matching engine to the same log data.

*RQ2: Will the system simplify the process of detecting compromised host in the network compared to manual inspecting each alarm?*

Even after removing some of the bad signatures searching log data with a large signature set will often generate very large numbers of alarms. Manually inspecting each alarm to detect compromised computers requires a lot manual work and is sometimes practically impossible. The proposed system will quantify the risk associated with each alarm and calculating an aggregated risk score for each host in the network. By doing this it should no longer be necessary to inspect each alarm to identify compromised hosts.

The hypotheses is that using the proposed system it should be possible to identified compromised hosts only be looking at the hosts with the highest risk score and the alarms that represents the highest risk.

The hypotheses will be tested by applying the proposed system to the same log data that was used for RQ1 and try to manually verify if the hosts with the highest risk score are actually compromised. A limitation with the proposed test is that it will be difficult to verify how good proposed system is beyond the fact that it can detect some compromised computers. It will not be possible to test if all compromised hosts were identified or what percentage of compromised hosts was identified. This is related to general challenge of identify ground truth in intrusion detection research(Sommer & Paxson, 2010b).

# 2 Background

The intrusion detection literature is vast and it can be challenging to identify the most relevant papers. This section starts with looking at intrusion detection in general and tries to frame it as an information retrieval problem. The second chapter follows up by looking more specifically on the problems of applying machine learning techniques in an adversary environment.

The proposed botnet detector is design mainly to detect botnets based on the IP-address and domain names they are using. In the section "2.3 Botnet detection based on network resources" different aspects of modern networks are discussed and how these influence the effectiveness of the suggested botnet detector.

In the last section the recent research on instruction detection is reviewed. The review is limited to intrusion detection based on netflow and passive DNS.

## 2.1 Information Retrieval

Information retrieval is a broad term used to describe many problems related to searching large amounts of data. This section discusses aspects of the information retrieval literature and how it relates to network based intrusion detection.

The main goal of network instructions detection systems is to detect if hosts on the network are compromised. A network based intrusion detection system typically achieves this by searching the network traffic for suspicious activity. Based on this search the system estimated the likelihood that hosts on the network are compromised. Using terminology from the field of information retrieval, this can be considered a classification problem. In a classification problem one is given a set of classes and the goal is to determine which classes a given object belong to. Casting network intrusion detection as a classification problem the classes would typically be: secure or compromised and the network traffic from each host the 'object' that represents the host in the classification algorithm. Again using information retrieval terminology, we can categorize the classification technique used in intrusion detecting based on the level of automation. (*Introduction to Information Retrieval [Hardcover]*, 2008)

- Manual classification
- handcrafted rules
- machine learning-based classification

### 2.1.1 Manual classification

Manual classification or inspection of network traffic can give a deeper insight into the workings of the network and what it is used for. The problem is that it is very labor intensive and it is difficult to scale to the massive amount of traffic that is normal in today's network. Manual inspection is therefore not practical as a general intrusion detection technique, but can be an important technique for getting insights into what is going on when investigating alarms generated by some of the more automated techniques.

### 2.1.2 Handcrafted rules

Handcrafted rules are one of the most common techniques in use today. The main reason is that handcrafted rules have good scaling properties and can be used to inspect enormous amount of network traffic very efficiently. In information retrieval terminology, this called standing queries. Figure 7 below is an example of a typical handcrafted network intrusion detection rule; it is a rule for the popular open source network intrusion detection system SNORT.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139
  flow:to_server,established
content:"|eb2f 5feb 4a5e 89fb 893e 89f2|"
msg:"EXPLOIT x86 linux samba overflow"
reference:bugtraq,1816
reference:cve,CVE-1999-0811
classtype:attempted-admin
```

Figure 7 Example of handcrafted snort signature

Intrusion detection based on handcrafted rules that specify malicious activity is in the intrusion detection literature often called misuse detection systems. Typically, a misuse detection system contains two major components (1) a language for describing known techniques (called misuse signatures) used by attackers to penetrate the target system, and (2) monitoring programs for detecting the presence of an attack based on the given misuse signatures. (Lin, Wang, & Jajodia, 1998). What is considered a hand crafted rule in information retrieval terminology is often called a signature in the intrusion detection literature.

One problem with applying handcrafted rules to intrusion detection is that both normal traffic and attacks change over time and keeping the signatures up to date becomes challenging. The gradual changing over time leads to large and complex rule sets. For example, Snort has two popular commercial rule sets, one from Source Fire[2] and one from Emerging threats[3]. They contain respectively about 13 000 and 20 000 signatures. The problem of keeping these handcrafted signatures sets up-to-date will probably increase as the use of internet continues to increase in the coming years and with it the scope of cybercrime. The web page av-test.org provides some statistics

[2] http://www.sourcefire.com/
[3] http://www.emergingthreatspro.com/

on the number of total classified malware sample that illustrates this problem.

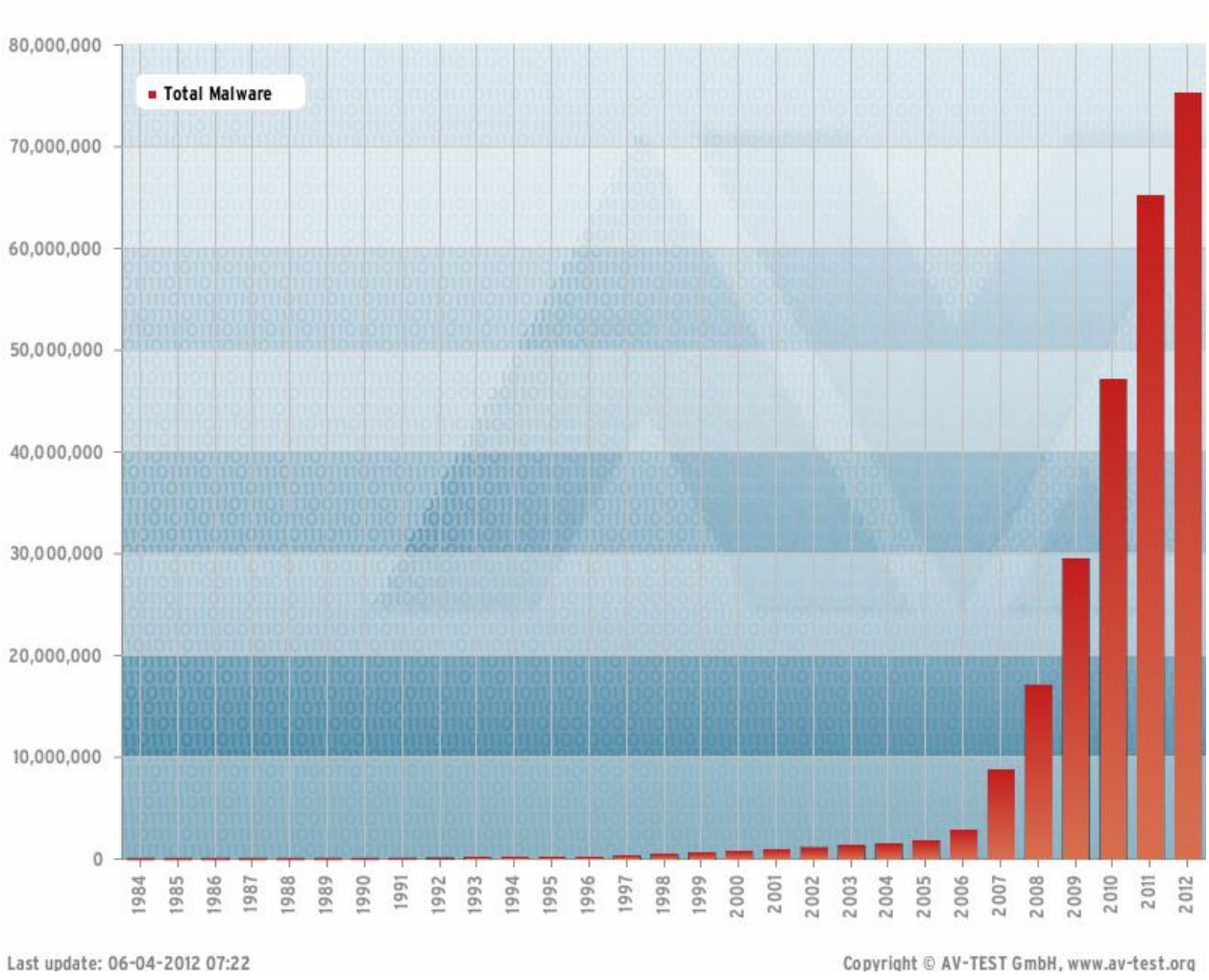



Figure 8 Illustrating how total number of classified malware has been increasing

Simple blacklists of malicious domains and IP-addresses can also be considered handcrafted signatures with a simplified rule language. The simplified rule language has the advantage that it easier to automatically share and integrate from multiple sources. Simple blacklists are also easier to automatically generate, using for example honeypots.

### 2.1.3 Machine learning

Apart from manual classification and handcrafted rules, there is a third approach, namely machine learning-based classification. In machine learning, the rules are learned automatically from training data. Machine learning based systems seem to have limited success in operational environments(Sommer & Paxson, 2010a)

A typical supervised machine-learning algorithm learns a classifier function based on the statistical properties of a labeled training dataset. The learned classifier can then predict which class new instances belongs to. For network intrusion detection the classifier can typically classify traffic as either malicious or benign or a host as secure or compromised. For most binary classification

algorithms, there is a requirement that a representative selection of instances from both classes must be contained in the training data. Often when applying machine learning to intrusion detection it is not possible to get a complete training set and the algorithm will often only have access to examples from one class.

The problem of learning from a classifier based on examples from only one class is called a one-class classification problem[4]. This problem is considered fundamental harder than the standard classification problem(Khan, Madden, Coyle, & Freyne, 2010). One-class classification is related to anomaly detection which is defined as the problem of finding patterns in data that do not conform to expected behavior (Chandola, Banerjee, & Kumar, 2009). Anomaly detection is also often referred to as outlier detection, where a outlier is: "an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism" (Hawkins, 1980). An Intrusion detection system based on anomaly detection learns a notion of what is normal activity and then reports deviations from that profile as alerts. The underlying assumption of such a system is that malicious activity exhibits characteristics not observed for normal usage (Denning, 1987)

From these definitions, we see that anomaly detection can be considered a one-class classification problem that is only using examples from the benign class. It is possible to gain a simple intuition of why one-class classification is a harder than normal classification problems from the illustrations in figure 9 below. In a standard learning problem, the learning algorithm is given a representative selection of examples from both classes and the learning algorithm is able to learn a boundary between the classes.

- [4] Other names for one-class classification is: "Concept-learning in the absence of counter-examples", Anomaly detection or "novelty detection"

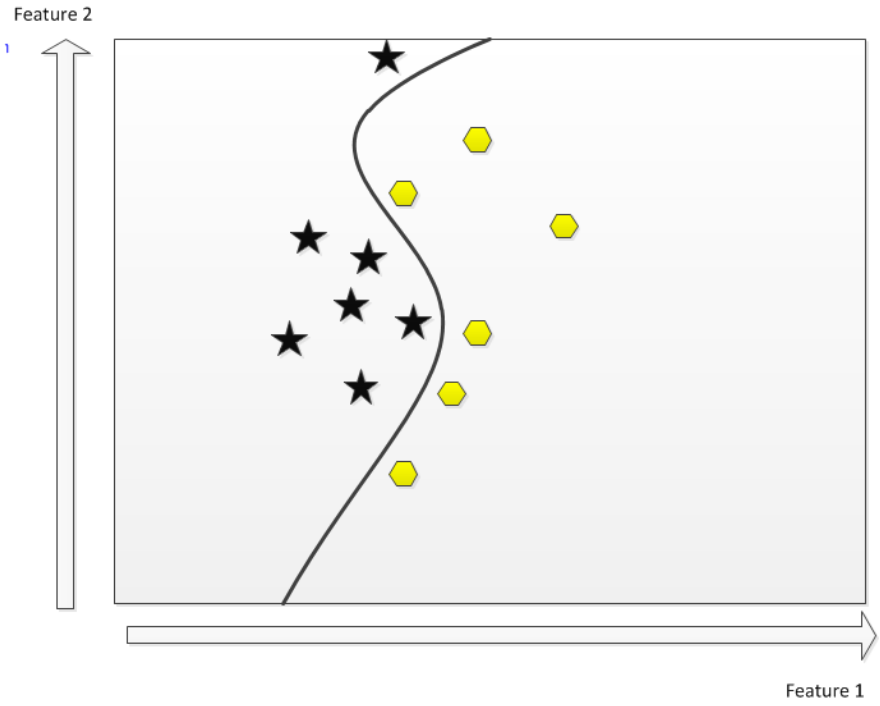



Figure 9 Example of the binary classification with two features

When the learning algorithm is given only examples from one of the classes, it is more difficult for the algorithm to decide where to draw the class boundary. This is illustrated in figure 10 below. In the figure to the left, the learning algorithm is presented examples from only one of the classes and learns a boundary based on the available training examples. In the figure on the right the examples from the other class is plotted as well. We can see that the algorithm misclassifies some of the examples. The underlying reason for the miss classifications is that it is very difficult for the learning algorithm to know where to draw the class boundary without counter examples from the other classes.

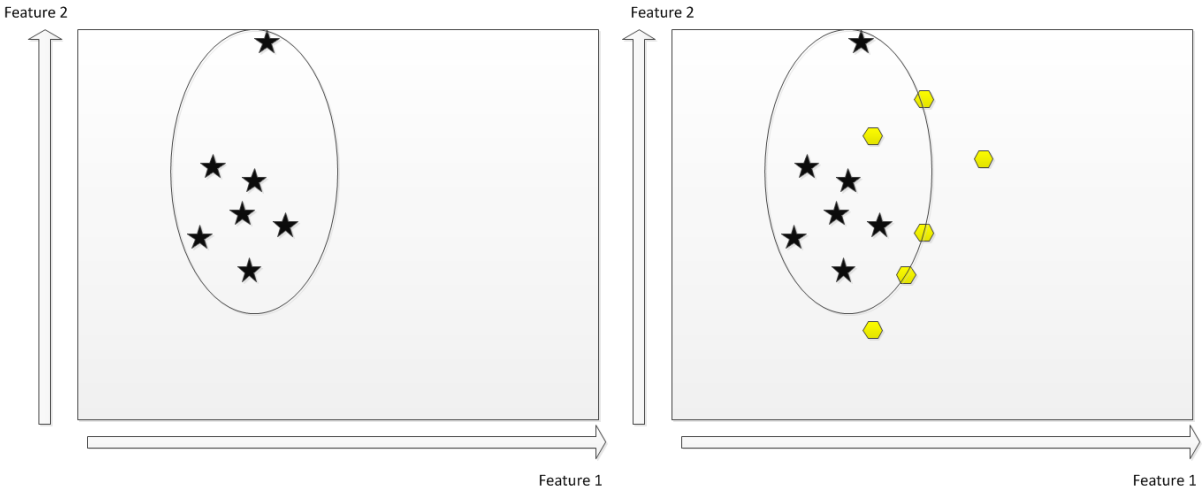



Figure 10 Example of classification with only examples from one classanomaly detection

From example in figure 10, we can get the intuition that a machine learning based intrusion detection system stands a better chance of being successful if the training dataset includes a selection of representative examples from both malicious and benign traffic.

Even though it seems obvious that anomaly based detection systems are sub optimal they are the most popular strategy employed in network intrusion detection research. One reason for its popularity might be the problem of creating and maintaining a good training dataset. Benign traffic data is often not shared because of privacy concerns and malicious traffic is not shared for security reasons.

Sommer et al criticizes the anomaly detection strategy in their paper *Outside the closed world: On using machine learning for network intrusion detection* and argues that the domain of network intrusion detection exhibits particular characteristics that make the use of anomaly detection particularly hard. They identify 6 characteristics:

1) The need for outlier detection,
2) High cost of classification errors,
3) A sematic gap between detection results and their operational meaning,
4) The enormous variability of benign traffic,
5) Challenges with performing sound evaluation,
6) The need to operate in an adversarial setting

The problem of *outlier detection* is related to the fact that real world network traffic from a heterogeneous network exhibits immense variability even within its most basic characteristics such as bandwidth, duration of connections etc. An anomaly system must per definition report all deviation from its profile of normality as a potential intrusion and experience has shown that legitimate activity often can generate many anomalies. Changes in network setup, the addition of a new server or client, a new popular internet service or all attendees of a meeting downloading the project files at the same time are example of legitimate activity that easily creates anomalies.

*The high cost of error* is related to the fact that every time the system generates an alarm, it will require the attention of an operator. A typical alarm from an anomaly detection system can be “unusual increase in outbound traffic”. It will take an operator considerable amount of time to figure out the underlying cause of the alarm to determine if it is malicious activity or not. The amount of traffic an NIDS inspect and the cost of verifying alarms make it critical that the false positive rate of a NIDS is very low and this is very difficult to achieve because of the high variability in the network traffic.

Sommer et al. reference to the *sematic gap* is based on the notion that an anomaly based intrusion detection systems detects anomalies and not attacks and it is up to the operator of the system to investigate the activity and decide if the anomaly is malicious or benign.

According to Sommer et al these characteristics are the underlying reasons for the notion in the information security community that anomaly detection systems create to many false positives to

be useful. It is also believed to be the reason that most of the machine learning based systems have had limited operational success.

Another aspect that is not discussed by Sommer et al is that every time the system triggers an alarm a human operator must investigate the threat. In this process the operator will typical learn a lot about the threat. A successful intrusion detecting system must make it easy to take advantage of the knowledge the operator gained during the investigation. Simple hand-crafted rules are simple ways to encode this knowledge back into the IDS. In contrast, when a machine learning based approach is used it is often difficult for the human operator to directly teach the system. Often when supervised learning algorithms are used, the only way to teach the system is to represent the new examples to the learning algorithm and hope that the algorithms find statistically significant attributes that can be used to classify the new attack. This indirect form of teaching is often not satisfactory. In the case of an attack, the operator will want some guaranties that the detection system will detect the attack in the future.

Another advantage of simple rules over machine learning is that they are easy to share across organizations compared to for example statistical classifiers. Considering the importance of sharing threat intelligence this might be another important underlying reason for the low usage of machine learning based intrusion detection system. Such systems might perform very well for a limited dataset or period when the authors put a lot of effort to encode and select the correct features for the learning algorithm, but as the threat gradually changes the system performance decreases.

All these challenges do not suggest that it is impossible to detect attacks with anomaly based network intrusion detection systems. Certain types of network attacks create very distinct network anomalies. Examples are scanning, denial of service and worm propagation, for these activities there are a number of anomaly based detection mechanisms that has been shown to work efficiently. An example of a successful commercial product based on these techniques is Arbor peakflow[5]. Botnet command and control traffic is an example of a malicious activity that is easier to make very similar to normal traffic and therefore difficult to detect with anomaly detection systems. There are for example few statistical differences between a malicious bot polling a Command and control server and a legitimate software agent polling a server for software updates.

## 2.2 Intelligent threat agent

A central challenge when applying machine learning to intrusion detection is that the source of the error is not just random variations, but an intelligent adversary working actively to defeat detection. This adds an extra dimension of complexity to the problem of measuring the effectiveness of the algorithm (Halpern & Tuttle, 1993).

Creating a machine learning technique that produces good results on present day malicious activity does not mean much because malicious activity evolves. Most intrusion detection techniques so far

[5] Arbor Peak flow http://www.arbornetworks.com/peakflowsp

have been like pesticides that do nothing more than create a new, resistant strain of bugs. The important question is: will the algorithm work even if the threat agent knew exactly how it worked?

A machine-learning algorithm claiming to be efficient in the domain of intrusion detection must therefore show that it can detect intrusions even if the adversary knows how the algorithm works. It must at least provide convincing arguments that is it very difficult for an attacker to avoid detection. If this is not the case the algorithm will probably be inefficient as soon as it gets popular. For example when checksum based antivirus detection became popular malware authors started to use more and more on polymorphic code and thus defeating the checksum based detection algorithms.

Because most errors are not random the design and assessment of intrusion detection algorithms requires a deeper understanding of the threat agent and his intentions. Assessing the effectiveness of a detection algorithm only using the adversaries' current techniques will most likely provide overoptimistic results because new attack techniques will not be random variations of existing attacks, but evolved versions designed to defeat the detection mechanisms.

Most cyber criminals are businessmen and their goal is to make money. Breaking into computers and recruiting them into botnets is only a mean to that end. If we get very good at detecting today's attacks, cybercriminals will try to design new ways of breaking into computers. The only way we can defeat them is if we can make it very difficult to make money on cybercrime. This would then over time probably lead to fewer cybercriminals, and less cybercrime. There are of course many important aspects to such a strategy; building better and more secure software, introducing better and more efficient international law enforcements and so on, but this paper is only concerned with the role of intrusion detection. If we could find effective and efficient methods for detecting these cybercrime operations that work over time, we could probably stop most of the attacks very early and it would be difficult to make money on cyber-attacks. The challenge is finding good general detecting methods that will work even when facing an intelligent opponent. In machine learning terminology this is sometimes called adversarial machine learning (Tygar, 2011)

Developing strategies in the face of an intelligent adversary is something military organizations have been doing for centuries. It therefore seems appropriate to look for inspiration from military strategy when designing system intended to fight cybercrime.

An important part of military strategy is intelligence, which is concerned with information collection and analysis to provide guides and direction in military planning. The process used by US and NATO forces today is called intelligence preparation of the battlefield (IPB). The original description of the method is found in US Army Field Manual 34-130 Intelligence preparation of the battlefield.

*"IPB is the best process we have for understanding the battlefield and the options it presents to friendly and threat forces."*

The IPB process is defined as a continuous process consisting of 4 steps:

1. Define the battlefield environment.

2. Describe the battlefield's effects.
3. Evaluate the threat.
4. Determine threat COAs.

The IPB process was developed to support kinetic military operations and it's not clear that is a useful approach to conflicts on the Internet or that it is useful against economically motivated criminals. One of the challenges of applying the IPB process to cyberspace is finding analogs for kinetic concepts such as terrain and weather. A simple analog for terrain is the parts of the IT systems that are not possible to change in the time frame of operation. Hanseth et.al has proposed calling these more slowly changing parts of information systems the Information infrastructure(Hanseth, Monteiro, & Hatling, 1996).

DARPA has been sponsoring various research efforts to test if the IPB process can be mapped to cyber defense. On such effort is the The Information Battle space preparation experiment(Moore, Kewley, Parks, & Tinnel, 2001). The goal of the experiment was to test if an adopted IPB process could improve detection, prevention and prediction of network attacks. The main conclusion of the experiment was that the adopted IPB process in fact seemed to improve all three: detection, prevention and prediction. When discussing why the IPB process allow for better prediction of enemy course of action (COA) they conclude with the following

*"We judge that this structured process encourages the human to think through all aspects of hostilities in a methodical way which allows him or her to identify potential COAs more accurately. IPIB takes a much more comprehensive view of adversary goals, vulnerabilities and attack axes to determine EnCOAs"*

The two key concepts for predicting how an adversary will act seems to be understanding the adversary's goals and how the environment creates opportunities and limitations for reaching those goals. For example in a traditional real world IPB process it would probably be natural to analyze the weather's impact, for example on visibility. Analyzing this in a systematic way one might come to the understanding that bad weather presents an opportunity for the adversary to conduct an operation in secrecy. Predicting this with some probability allows for planning a defensive strategy for such a situation. Comparing this to cyberspace there is no clear analogy for weather conditions, but there are clearly concepts that affect the visibility from a NIDS point of view. One example of this is cryptography. Switching from HTTP to HTTPS makes it possible for an adversary to hide the URL used in command and control traffic from a NIDS sensor point of view. In fact using cryptography will render all full content NIDS signatures that relay on data above the IP layer in the TCP/IP model useless. This example implies that it important to take into consideration how the environment (IT infrastructure) creates opportunities. For example designing machine learning algorithms that depend on full content data present the opportunity for the adversary to avoid detection with the use of cryptography. This implies that a machine learning algorithm that hopes to be successful over time in the face of an intelligent adversary must not depend on concepts that can easily be avoided by an adversary.

The biggest contribution from the IPB process seems to be the insight that a systematic method for analyzing how the environment create opportunities both for adversaries and defenders it important when facing and intelligent adversary.

## 2.3 Botnet detection based on network resources

In the previous section, it was identified that understanding the IT infrastructure and the opportunities it presents for the adversary is important. This section will discuss how the modern networks affects a NIDS ability to detect intrusions and an argument will be made that most botnets can effectively be tracked by the network resources they are using.

### 2.3.1 Stateful firewalls

Today most home networks and corporate networks are behind a stateful firewall or Network address translation (NAT) router. This has affects how adversaries can attack and secure access to compromised computers. Stateful firewalls forces attack related traffic to originate from the inside of the intranet network.

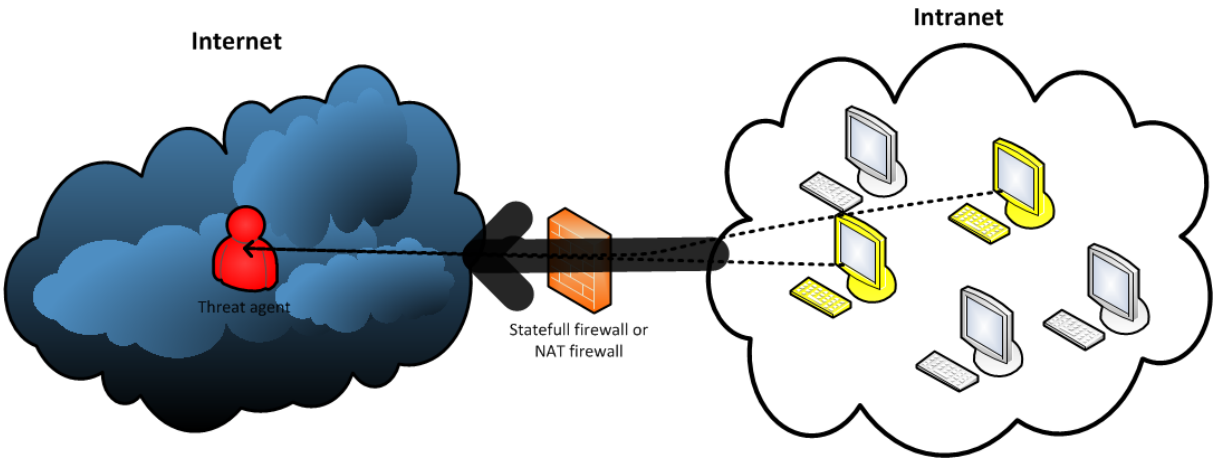



**Figure 11 Illustrates that the introduction of firewalls forces adversaries to install bots that poll for new instructions to ensure continued access**

The fundamental change with the introduction of firewalls is that the bot has to be preprogrammed with the location of the command and control server. This forces the location of command and control servers to stay the same between each time the bot polls for new instructions. If the location of the command and control server changes when the bot is offline, the attack er will lose control of the compromised computer. Especially in large botnets there will always be some bots that are offline for various reasons, this makes changing the command and control very difficult for the attacker. This fundamental limitation of botnets makes them easy to track. As the size of the botnet increases, it will get increasingly difficult for an attacker to keep the command and control lookup mechanism secret.

### 2.3.2 Command and control topologies

Cyber criminals are well aware of the fact that the command and control server is a critical vulnerability. As a countermeasure they have developed a series of different command and control communication topologies, making it more difficult to track the botnets(Ollmann, 2009). The

different topologies can be compared on how difficult they are to setup and maintain, how robust they are against takedown, and how difficult they are to track. Example topologies are single server, multi-server, hierarchical and peer to peer.

The simplest topology is a single server topology. This topology is the simplest to setup, but is also the simplest to track and shut down. The next topology is multi-server topology which adds more servers. This topology requires more resources to setup and maintain because of the introduction of multiple server, but it also makes it more difficult to track and shut down.

Because most network equipment such as routers and firewalls can track and block traffic based on IP-addresses even the most advanced IP-address based topologies are relatively easy to track and take down. As a countermeasure cyber criminals have developed strategies that enable them to resolve the IP-address of the servers dynamically thus making them less dependent on having command and control server with static IP-addresses.

The domain name system is the standard mechanism for resolving IP-addresses on the internet today. By using the domain name system a botnet can potentially become more resilient to server takedowns. If a server is taken down a new one can be created and the domain name record updated to point to the new server. The use of domain names creates a new problem for cyber criminals; the domain is now a new single point of failure. To mitigate this problem they have developed even more advanced command and control topologies. Examples of such topologies are IP flux, Domain flux, domain wildcarding and Random domains[6]

Criminals can in theory depend on other services than the domain name system for resolving the IP-address of the command and control server. There have been examples of bots using for example closed Google groups or Twitter channels. But historical data seems to indicate that most cyber criminals prefer to use the domain name system. There are some properties that seem to make the domain name system better for cybercriminals comparing to internet services like Twitter and Google groups.

The domain system is a very large distributed system with many independent domain name registrars[7]. Most of these registrars base their incoming on monthly fees from registered domains. Terminating domains because of illegal activity reduces their income and at the same time the cost of the illegal activity is external to the domain registrar. This creates a situation where some registrars are not very willing to terminate domain registrations because of illegal activity. Some might even be reluctant to terminate domains when given proof that they are used for malicious activity. Others have very little resources to deal with misuse cases. Because of this it can be

[6] These topologies are discussed in (Ollmann, 2009) and will not be described in detail her.

7 domain name registrar is an organization or commercial entity, accredited by both ICANN and generic top-level domain registry (gTLD) to sell gTLDs and/or by a country code top-level domain (ccTLD) registry to sell ccTLDs; to manage the reservation of Internet domain names in accordance with the guidelines of the designated domain name registries and to offer such services to the public. (source: Wikipedia.org)

challenging to take down domain names used by command and control servers. This is a very attractive property for cyber criminals.

This is in contrast to using an Internet service like Twitter or Google groups where a single organization is in control. If their service is associated with botnet activity they can easily be blacklisted or lose credibility as a secure service. The cost of botnet activity is therefore more likely to be internal to the business running the service and they might therefore be more motivated to stop the activity. A botnet depending on a service like Twitter is therefore facing the risk of faster removal, which can leave the botnet without a command and control server and effectively disabling the botnet.

Another advantage of using the domain name system is the share number of registrars. If one registrar becomes really good at detecting and shutting down domains used for malicious activity it is very easy to switch to another registrar. In the case of an Internet service, for example Twitter, if they get really good at detecting and removing accounts used for malicious activity there is no other place to go.

Even though botnets based on the domain name systems can be hard to takedown, they are relatively easy to track from a NIDS sensor. The reason for is that all DNS traffic will normally pass a NIDS unencrypted.

### 2.3.3 Cryptography

Full content signatures make it possible to detect characteristics of the botnet traffic itself. This will often make it possible to detect all botnets that share the same botnet kit. For example, the snort signature below can reliably detect all traffic related to certain version of the Zeus botnet kit.

**Table 1: Example snort signaure**

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET
CURRENT_EVENTS Zeus Bot / Zbot Checkin (/us01d/in.php)";
flow:established,to_server; content:"GET"; nocase; http_method;
content:"/us01d/in.php"; nocase; http_uri;
reference:url,garwarner.blogspot.com/2010/01/american-bankers-association-
version-of.html; reference:url,doc.emergingthreats.net/2010729;
classtype:trojan-activity; sid:2010729; rev:5;)
```

The problems is that if the cyber criminals become aware of the fact that their command and control traffic is primarily tracked based on full content signatures they can enable encryption and that will render the signatures useless. Take the example signature above, the creators of Spyeye kit has already created a crypto plugin that makes it possible to encrypt the botnet command and control traffic.

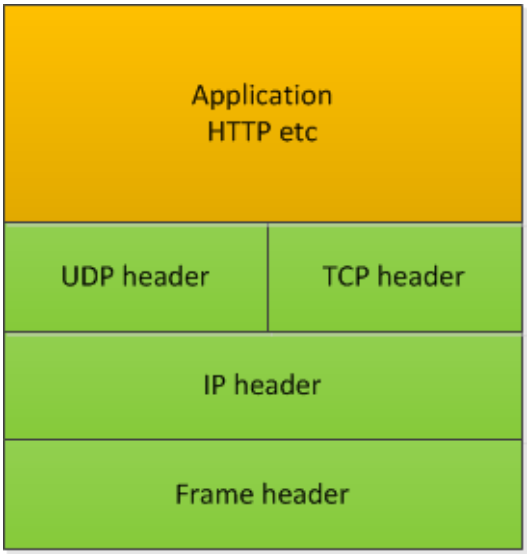



**Figure 12 The Orange color illustrates that all data in the application layer of the TCP/IP stack can potentially be encrypted, and detection based on it is easy to avoid.**

This fundamental weakness of full content signatures in relation to botnet tracking does not indicate that full content signatures are useless in all cases. For example the exploit of a vulnerability is generally not possible to encrypt and full content signatures can be a very powerful tool in detecting such an attack.

Even though it would be easy to encrypt botnet traffic, experiences from popular botnets indicate that many of them are not encrypted. One reason for this might be the added complexity of using cryptography. Even though full content signatures still are an efficient method for some botnet detection, it is probably not a good idea to design a system that relies on it because the mechanism is so easy to defeat for botnet operators.

### 2.3.4 IP-adresses

For hosting command and control servers, cyber criminals can choose between renting a server or hijacking and using a compromised server. Renting their own server has the advantage that it's simpler; they don't have to first compromise an Internet facing server for command and control purposes. Another advantage is that it's often more stable, a compromised computer might get cleaned up and if the bots are not programmed with redundant s servers the attacker might lose control over the bot net.

It's very important for cyber criminals are to remain anonymous to avoid law enforcement. If the service providers the criminals are renting servers from are cooperating with law enforcement they

might provide traffic logs or a ghost image of the rented server. And in such a situation the anonymity of the criminals is in danger. For this reason cyber criminals prefer to rent servers from countries with weak cyber legislation and from service providers with a reputation for not cooperating with law enforcement. These service providers are often called bullet proof hosters because they normally ignore takedown requests and allow criminals to conduct their business uninterrupted. Over the last couple of years a long list of bullet proof hosters has been identified and taken down. The list below is examples of bullet proof hosters from the last couple of year.

- Russian Business Network (or RBN), taken down in November 2007 [4]
- Atrivo/Intercage, taken down in September 2008 [5]
- McColo, taken down in November 2008 [6]
- 3FN, taken down by FTC in June 2009 [7][8][9]
- Real Host, taken down in August 2009 [10]
- Ural Industrial Company, taken down[*citation needed*][11]
- Group Vertical, taken down in Oct 2009[12]
- Riccom, taken down in December 2009 [13]
- Troyak, taken down in March 2010 [14]
- Proxiez, taken down in May 2010 [15][16]
- Voze Networks, taken down in February 2011 [17]
- Ecatel, still operating [18]

**Figure 13 List of know pullet proof hosters**[8]

This means that there are certain IP-address spaces that are more likely to be related to cybercrime and botnet command and control servers. The challenge for cyber criminals is that using bulletproof hosting protects their anonymity, but it makes it much easier to detect the botnet. Traffic to or from their IP-address space could be considered very suspicious. The fact that many botnet operate from within a small IP-address space is a big advantage because by tracking the IP-address used by known botnet command and control server new botnets that operate within the same address space can be detected.

One of the solutions cybercriminals have come up with to avoid the problem is a layered botnet topology. They use compromised computer to proxy the traffic to a central command and control server that is hosted by a bullet proof hoster. The usage of hijacked server as reverse proxies defeats some of the IP-address based botnet tracking because it is not possible to determine if the traffic is botnet related if the traffic goes to a hijacked server.

Even though the use of hijacked servers as reverse proxies isis making IP based botnet tracking much harder, the usage of the technique seems to be limited. The reason is probably that maintaining a large pool of hijacked server for reverse proxies requires a lot of resources from the botnet operator.

[8] http://en.wikipedia.org/wiki/Bulletproof_hosting, last visited 10.06.2012

### 2.3.5 Domains

When selecting domains for their command and control server cyber criminals can choose between registering their own domains or hijacking legitimate domains. The tradeoffs associated with this choice are similar with that of renting or hijacking a command and control server. Renting is easier because it is not necessary to hijack a legitimate domain. The biggest disadvantage for cyber criminals when registering their own domains is that they might leave clues that makes it easier to identify them.

### 2.3.6 Retrospective analysis

The DNS and netflow logs even from big networks are small enough that they can be stored for years. If a new botnet is detected, old activity logs can be searched and an overview of compromised computers can be found relatively quickly. The possibility to perform this kind of retrospective analysis is a powerful feature of botnet tracking based on domain name and IP-addresses.

At first glance it might seem that this kind of retrospective analysis is of little value because if the computer has already been compromised for some time an argument can be made that the damage has already been done. It is reasonable to assume that as soon as the computer was compromised some of the sensitive information on the computer was stolen, such as credit card information, usernames and passwords stored in the web browser.

If the infected computer contains other sensitive information or is connected to other systems with sensitive information, it will normally require a manual targeted attack from the owner of the botnet to get the information. After the initial infection, there will therefore be a race between the attacker and the victim's organizations security team. The team defending the network will have to clean up the compromised computer before the attacker is able to perform a targeted attack and do more damage.

Experience has demonstrated that attacks are often detected sometime after they are performed. I n such a situation retrospective analysis is crucial to get an overview of the attacks. We know that cyber criminals have developed a black marked for buying and selling cybercrime services,[9] it is therefore not unreasonable to assume that access to computers that might hold sensitive information is sold to individuals that for example specializes in industrial espionage. Because of scenarios likes this it will always be important to detect compromises as soon as possible and get the computers cleaned up.

Given the importance of retrospective analysis, sharing threat intelligence with the rest of the security community in the form of domains and IP-addresses seems a good idea to help improve the overall ability to detect cybercrime activity.

[9] http://us.norton.com/cybercrimeindex/blackmarket.jsp

## 2.4 Related work

The system for detecting botnets proposed in the next chapter is based on tracking botnets based on IP-addresses and domain names. This section will review the current literature on using netflow and passive DNS for detecting and tracking botnets.

### 2.4.1 Tracking botnets using passive DNS

The most recent paper that studies how botnets and malware activity in general can be detected using passive DNS is *Exposure: Finding malicious domains using passive dns analysis*(Bilge et al., 2011). The exposure system can be considered a classical machine learning system. First training data is labeled using popular blacklists and white lists. For blacklists spyeyetracker, malwaredomainlist etc are used and for whitelisting the top 1000 most popular web pages are used, based on the assumption that the most popular pages on the internet are professionally managed and not malicious. The authors then extracts a set of features they believe will separate malicious and benign traffic, and a machine learning algorithm is used to train a classifier on the available training data. Bilge et al had access to traffic from central DNS server on the internet that received about 1 million queries per minute. During the two and a half month trial they monitored approximately 100 billion DNS queries. Unfortunately they had a problem processing that much data. They therefore had to filter out some domains. The authors removed requests going to the domains on the Alexa top 1000 domain list, under the assumption they were benign. The next filtering step they performed was to remove all domains that were older than 1 year. The argument was that most malicious domains are disclosed after a short period and domains older than 1 year are therefore most likely benign. These two filtering steps removed about 50 % of the traffic

**Table 2 List the features used by exposure to classify domains**

| Feature Set | # | Feature Name |
|---|---|---|
| Time-Based Features | 1 | Short life |
| | 2 | Daily similarity |
| | 3 | Repeating patterns |
| | 4 | Access ratio |
| DNS Answer-Based Features | 5 | Number of distinct IP addresses |
| | 6 | Number of distinct countries |
| | 7 | Number of domains share the IP with |
| | 8 | Reverse DNS query results |
| TTL Value-Based Features | 9 | Average TTL |
| | 10 | Standard Deviation of TTL |
| | 11 | Number of distinct TTL values |
| | 12 | Number of TTL change |
| | 13 | Percentage usage of specific TTL ranges |
| Domain Name-Based Features | 14 | % of numerical characters |
| | 15 | % of the length of the LMS |

Bilge et al has proposed many new and innovative features but some of proposed features will be problematic in certain settings. The time-based features for example have the problem that they depend on a time series algorithm that only works if there are many requests for the analyzed

domains Because of this, the authors decided to only include domains that in total had more than 20 quires directed toward them. This led the authors to exclude 4.5 million domains from the dataset. After removal of 4.5 million domains the dataset contained only 300 000 domains. Table 3 below summarizes the detection rates that Bilge et al reports on the filtered dataset.

**Table 3 The reported accuracy of Exposure (AUC= Area Under the ROC Curve)**

| | AUC | Detection Rate | False Positives |
|---|---|---|---|
| Full data | 0.999 | 99.5% | 0.3% |
| 10-folds Cross-Validation | 0.987 | 98.5% | 0.9% |
| 66% Percentage Split | 0.987 | 98.4% | 1.1% |

The results seem impressive, but analyzing the performance of the algorithm on less than 7% of the domains in the original dataset makes it difficult to evaluate the results.

In two situations, the Exposure system queries Google for information about a domain. 1) *To determine if an IP is used by a shared hosting service, we query Google with the reverse DNS answer of the given IP address. Legitimate web hosting providers and shared hosting services are typically ranked in the top 3 query answers that Google provides. This helps us reduce false positives.* 2) *for domains that are determined to be suspicious, we check how many times it is listed by Google. The reasoning here is that sites that are popular and benign will have higher hit counts.* Using Google, as an external oracle is problematic because it makes it very difficult to determine how good the select features and machine learning algorithm really are.

The usage of Time to live(TTL) features and searching for fluxing TTL values will also be problematic in some real world environments. This is problematic in cases where the DNS sensor will observe DNS traffic from caching DNS servers and not traffic directly from an authoritative DNS server. When a caching DNS server responds to a query it will subtract the number of seconds the DNS records has already been cached from the original TTL value. A sensor observing traffic from a caching DNS server will therefore observe fluxing TTL values even though the original DNS records TTL value is unchanged.

Another recent paper that looks at botnet tracking using passive DNS is: Building a Dynamic Reputation System for DNS(Antonakakis et al., 2010). This paper shares many characteristics with the previous paper. It describes a system called Notos that is designed to be able to dynamically assign reputation score to new domains. The premise of this system is that malicious, agile use of DNS has unique characteristics and can be distinguished from legitimate, professionally provisioned DNS services. The system uses similar features as the Bilge et al paper. In addition to a supervised learning algorithm similar to Exposure, Notos also uses a clustering algorithm for grouping similar activity.

Evaluating the Notos system using 9530 known bad domains and the Alexa top 500 most popular domains gave a false positive rate (FP%) of 0.38 % and a true positive rate (TP%) of 96.8%. When

extending the known good domains to 100 00 domains the detection rate dropped to a FP% of 0.6% and TP% 80.6%. The big drop in TP% might indicate that Notos has some problems determining if less popular sites are malicious or benign. This would limit the usefulness of the system as a standalone system. The authors conclude that the system it would be a best to use it in combination with other detection techniques.

In chapter 3.1 the paper introduces a practical and concise formal terminology for describing the relation between, domains, top level domains, second level domains, IP-addresses, zones, IP-networks, PGB routing prefixes, AS numbers. This terminology seems promising as a standard terminology when describing these relationships.

The detection systems Notos and Exposure are complementary to the system proposed in this paper. They use a limited set of blacklisted domains to learn statistical properties about malicious domains that enables them to detect new domains. The system proposed in this paper is design to be able to use a much larger and more unreliable signature set of malicious domains directly.

The strength of Notos and Exposure are that they can learn a statistical profile of malicious activity based known threats provided in blacklists. These techniques make it possible for Notos and Exposure to detect malicious activity even if the domains and IP-addresses are not blacklisted, as long as the activity is with in statistical profile of malicious activity. The disadvantage of these systems is that is difficult to guarantee what they will detect. For example traffic to domains and IP-addresses well known for their maliciousness can go unnoticed if they operate outside the statistical profile of malicious activity.

The proposed system in this thesis is complimentary to Notos and Exposure because the detection method is based on directly matching large signature sets with the activity log. This has both disadvantages and advantages compared to the statistical method used by Notos and Exposure. The disadvantage of the proposed system compared to Notos and Exposure is that it will be unable to detect malicious activity if the activity uses completely new domains and IP-addresses. The advantage of the proposed system is that it will always trigger an alarm if the domain or IP-address is known to be malicious. From this we see that the Notos and Exposure has complementary strengths and weaknesses compared to the system proposed here. A possible synergy is that domainsdomains flag as suspicious by Notos and Exposure can be used as input to the system proposed in this paper and by doing so improve the systems detection capability. In addition the system proposed here includes an algorithm for handling large amounts of alarms which could potentially improve the operational usefulness of Notos and Exposure.

### 2.4.2 Tracking botnets using netflow

Two often cited tools with regard to detecting botnets using netflow data is BotMiner (Gu, Perdisci, Zhang, & Lee, 2008) and Botsniffer (Gu, Perdisci, et al., 2008) and Botsniffer(Gu, Zhang, & Lee, 2008). These two systems are similar as they both utilize horizontal correlation across multiple host. The underlying assumption is that the system is monitoring multiple hosts infected by the same botnet and that these infected computers will behave in a coordinated way. In the last couple of

years there have been multiple law enforcement efforts to take out the large botnets; this has led to a trend where there are many small botnets instead of few large ones. This trend might be problematic for these detection techniques because they depend on correlating activity from multiple infected hosts from the same botnet. The likelihood of having multiple bots from the same botnet decreases drastically as the size of the botnets decreases.

Another tool that uses netflow is BotHunter (Gu, Porras, Yegneswaran, Fong, & Lee, 2007) this tool is designed to track the two-way communication flows between internal assets and external entities, developing an evidence trail of data exchanges that match a state-based infection sequence model. A web page www.bothunter.net has been created to support the continued development and distribution of the bothunter tools. An updated infection dialog model can be found on the web page. Dialog in this setting means stages in an attack.

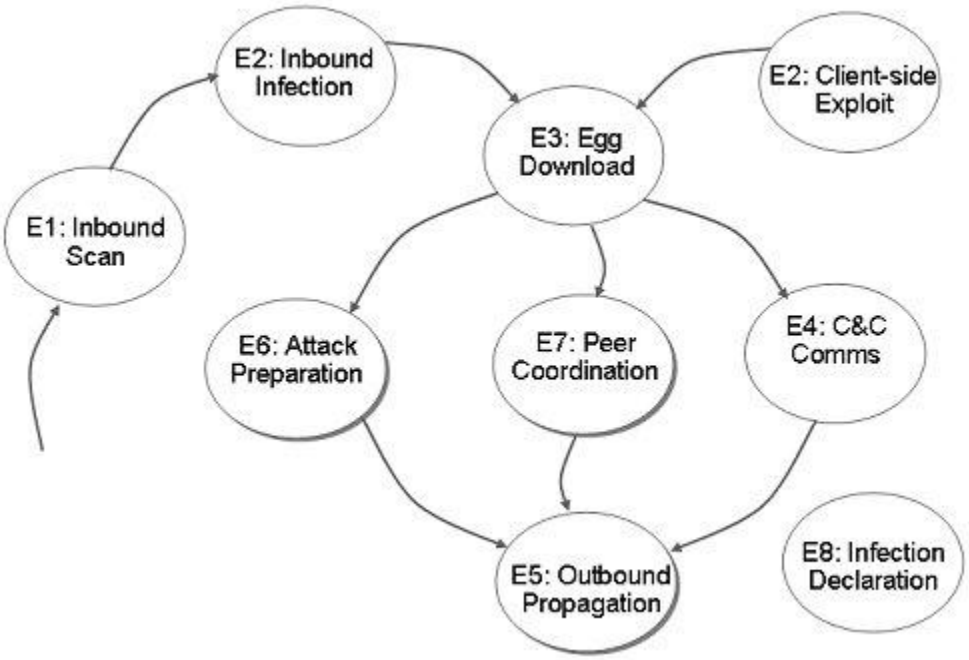



Figure 14 BotHunter's Infection Life Cycle Model[10]

Bothunter uses a dialog like the above to perform dialog sequence analysis, if a host has alarms from many of the states it is assumed to be infected. The system uses a modified version of snort to generate the dialog alarms.

[10] http://www.bothunter.net/about.html

# 3 Methodology

An overview of the proposed system is presented in chapter 4.4. The following chapters explain the individual components in more detail.

## 3.1 Quantifying signatures

This section discusses an algorithm for assigning a risk score to signatures. The risk score is a number that represent the quality of the signature. A high number means that the alarms generated by the signature are a good indication that the victim is compromised.

There are a many blacklists on the internet and a signature can often be found in multiple blacklists. The first section below discusses an algorithm for assigning a risk score contribution from each signature list. The second section discusses an algorithm for combing the risk score contribution from each signature list into a total risk score.

### 3.1.1 Risk score contribution

The different blacklists use different methods for gathering threat intelligence and creating signatures. Some of them get their signatures from web pages where community members report malicious IP-addresses or domains. Example of such a list is Malware Domain Lists. Some lists are generated by running malware samples in sandboxes and automatically extracting the command and control server. For some blacklists like Spyeye Tracker the collection strategy is not publicly known. Because the collection strategy varies, from blacklists to blacklists, so does the quality of the lists. Some blacklists are very reliable and provide signatures with very low false positive rate, other are inherently unreliable and provide little context information with each signature.

Some of the lists are freely available, some cost money and some are only available to members of certain security communities. For this thesis only freely available blacklist are used.

Different signatures can indicate different stages of an attack. Some signatures represent a botnet command and control servers and any traffic to it is a very strong indication that the client is infected. Other signatures indicate that the domain or IP-address is serving exploits and traffic to it does not necessarily indicate that the client is compromised. A network host is therefor much more likely to be compromised if he triggers a command and control related signature compared to triggering a exploit related signature.

When using large signature sets the intrusion detection system will typically generate a lot of false positives and signatures from low reliability blacklists typically generates the majority of the false positives. There is a danger that so many false positives are generated that important alarms can go unnoticed. Excluding large blacklists with lower reliability is not desirable either because they often contain some useful signatures. To be able to use multiple blacklists it is important to be able to quantify a risk score associated with each signature and put more weight on signatures from high reliability lists.

The botnet detection tool BotHunter (Gu et al., 2007) discussed in the previous chapter uses an detailed state-based infection sequence model and match signatures to this model. The problem with such an approach is that many of the easily available blacklists do not seem to provide enough context information to do this. For example, many of the blacklists do not provide enough information to determine if the signature represents command and control traffic or an exploit attempt. For this reason, it is difficult use and detailed infection model and simpler model is chosen here.

Each blacklist has a function that assigns a number between 0-100 to each signature. The number is the blacklists *risk score contribution* for the signature. This number represents the risk that a client is infected given that the alarm triggers on traffic from the client. 100 indicate a very high likelihood of being infected.

Two primary factors influence the signature risk score contribution.

1) The activity the signature represents
2) How reliable the signature produces alarms that are true positives

For example, the signatures from Spyeye Tracker represent command and control traffic from the SpyEye crimeware kit. If a host on the network is talking to a command and control server, it is a strong indication that the computer is compromised. The Spyeye Tracker also has a reputation of providing signatures with low false positive rate. Signatures from Spyeye tracker should therefore be assigned a high risk score contribution.

A custom function for assigning risk score to signatures that uses all available information can be developed for each signature list, but because most signature lists provide little or no context information, a simple general function will therefore suffice in most cases. The proposed algorithm for calculating the risk score for each signature uses the three following core concepts:

- Signature
- Tag

The Tag concept is introduced to make it easy to group signatures together. The tag concept has one important attribute and that is the tag severity, representing the severity of signatures that has the given tag. The severity is a number between 1 and 100 representing the likelihood that the computer is infected given that the computer triggers the signature. Example of tags can be botnet CC, drive by download, exploit kit or bulletproof hoster.

The source concept is introduced to make it easy to relate the signature and the tag to the source that is reporting it. The source concept has one important attribute and that is the source reliability, which tells the reliability of the source. The reliability is a number between 1 and 100 and represents the likelihood that the source is reporting a signature that does not trigger false positives.

The figure below is a relational model describing the relationship between signatures, tags and sources.

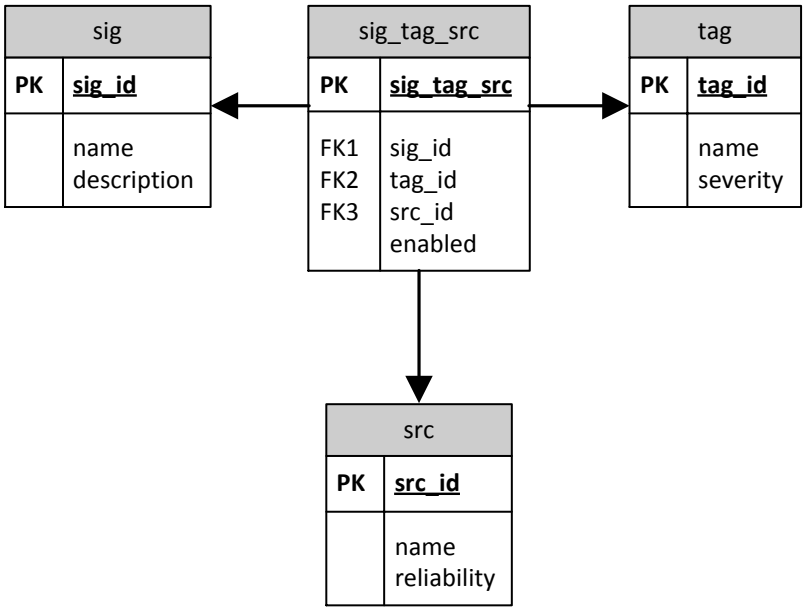



Figure 15: Relational model

The relational model illustrates that a signature can have multiple sources and each source can assign multiple tags to each signature.

When calculating the signature risk score a central concept is that the score should be proportional to the reliability of the source reporting the signature and the severity of the tag associated with the signature. The algorithm for calculating the risk score contribution for a signature with one source and one tag is therefore:

$$signature\_risk\_score\_contribution = source\_reliability * tag\_severity$$

Sometimes a source might report multiple tags on a single signature. For example, a source might tag some signatures with both the tag *zeus_botnet* and *botnet.* This double tagging is often done only to make it easier to organize the signature. From this it seems reasonable to conclude that the risk score contribution from a single source should be independent of the number of tags the source has given the signature. It therefore seems reasonable that the best approach is using the tag with the highest severity when calculating the signatures risk score from a single source. The algorithm for calculating the risk score contribution for a signature with one source and multiple tags is therefor:

$$signature\_risk\_score\_contribution = source\_reliability * \max(tag\_severity)$$

### 3.1.2 Total risk score

When multiple sources report a signature as malicious it seems reasonable that the likelihood that the domain is actual evil increases and therefore should the risk score associated with the signature also increase. For example if the domain *evil.com* is reported as malicious by both *Zeus_Tracker* and Malware Domain List the overall risk score should be increasing.

Intuition seems to indicate that the risk score contribution given by independent sources are additive. By this, we mean that a signatures total risk score is always increasing when more sources report it as malicious. If one source as given a signature a high risk score contribution and another source has given the same signature a low score the contribution to the total risk score from the last source should be small but positive.

It is also desirable that the signature risk score algorithm is design in such a way that signatures with low severity that are reported by many sources with low reliability rarely have a higher total risk score than a signature that is reported with a high severity from a single source with high reliability. This seems to indicate that the contribution to the total risk score from each source should be sub linear. An algorithm that seems to work in many similar situations is the Root Sum Square (RSS) algorithm. The total risk score for a signature is the root sum square of the risk score contribution from each source. Based on the intuition described so far an algorithm for calculating the total risk score can be constructed.

$$total\_risk\_score = \sqrt{\sum_{sources} risk\_score\_contribution^2}$$

The algorithm can be further mathematically formalized. To do that some basic terminology is necessary.

A set of tags is described with a capital T, a particular tag with a letter case t. The severity rating of a particular tag is given by the term $t_{severity.}$

A set of sources is described with a capital S, a particular source with a letter case s. The reliability rating of a particular source is given by the term $s_{reliability}$ and the risk score contribution is given by the term $S_{score\_contribution}$

A set of signatures is described with a capital R, a particular signature with a lower case letter. The risk score associated with a signature is given by the term $r_{score}$

The sets are narrowed down using sub scripts. For example T means all tags in the database, $T_s$ all tags associated with a particular source s and $T_{sr}$ all tags associated with a particular source s and a particular signature r.

Based on this terminology the risk score contribution from a single source can be expressed as following

$$s_{score\_contribution} = \max_{T_{rs}}\left(t_{severity}\right) * s_{reliability}$$

The total risks core associated with a signature described informally above can be mathematically expressed as following:

$$r_{score} = \sqrt{\sum_{S_r} \left(s_{risk\_score\_contribution}\right)^2}$$

It is important to note that the total risk score algorithm is only dependent on the risk score contribution. Each signature list can there implement its own algorithm for estimating the risk score contribution.

## 3.2 Whitelisting signatures

Because signatures are automatically loaded from many blacklists with varying quality, sometimes there will be bad signatures that can trigger a lot false positives.

For example, some blacklists are generated by running malware in a sandbox and automatically adding the command and control domains to a blacklist. This will generate false positives when the malware sample tries to contact a legitimate host to check if it has internet access. For example if a blacklist connects to google.com to check for internet connectivity, google.com can end up being blacklisted. In some cases, the malware might contact legitimate domains only to confuse malware researches and pollute blacklists. This is problematic because these bad signatures can generate so many alarms the alarm management software can stop working as intended.

Another problem is that an automatically generated signature can trigger on domain names or IP-addresses that are used internal on the monitored network. For example if the local IP-addresses 192.168.1.1 is in a blacklist and it is also used for something important on the monitored net this can create a lot of false positives. If the internal DNS server or internet gateway has a blacklisted IP-address it is possible to come in a situation where traffic on the network is generating an alarm and this will in most cases flood the alarm management system.

To be able to use these automatically generated blacklists it is therefore important to have good mechanisms for removing bad signatures.

The whitelisting method proposed here uses 3 signatures sets.

1. A black signature set with all signatures indicating malicious activity,
2. A white signature set with all signatures that are known to be non
3. An asset signature set with all local IP-addresses and domains.

A central component of the whitelisting mechanism proposed here is the risk score algorithm described above that assigns a score to each signature in a signature set. For the white and asset signature set the "risk score" indicates how confident we that the content it matches on is NOT malicious.

The whitelisting algorithm checks if the black signatures set overlaps with the two others. If a black signature is conflicting with a signature in the two other signature sets the relative score determines if it is going to be whitelisted or not.

If the black signature has a lower score it is blacklisted, if it has a lower score it not blacklisted. By designing the algorithm like this, we reduce the likelihood that a good signature in the black signature set is whitelisted by a bad signature in the white signature set.

To be able to deal with the different types of signatures efficiently some extra logic is required. This logic is described in the following sections.

### 3.2.1 IP-addresses

The system understands three types of IP-address signatures: single IP-addresses, IP-ranges and CIDRs. Because the number of signatures can be high, an efficient algorithm is needed to check if any of the blacklisted signatures matches any of the white listed. Because IP address ranges can be represented as number intervals the problem can be casted to a problem of finding overlapping intervals. There are many well-known Interval tree algorithms with good run time properties. Typically the query time is O(log n) where n is the number of elements in the interval tree.

The basic workings of the IP-address whitelisting algorithm is to first build an interval tree of all IP-addresses and IP-ranges in the whitelist and asset signature set and then query the interval tree for each signature in the black signature tree.

### 3.2.2 Domains

For domains simple string matching is enough in most cases but some extra logic is needed to deal with subdomains.

For example if google.com is whitelisted with a high risk score and sites.google.com is blacklisted with a low risk score it is desirable that sites.google.com is removed from the blacklist. The problem is that it is not always desirable to remove subdomains of whitelisted domains from the black signature set. For example if dyndns.org is whitelisted and evil.dyndns.org is blacklisted it is not desirable to remove evil.dyndns.org from the blacklist.

This problem is related to what is called public suffix or effective top-level domain (eTLD). According to Mozilla: *"public suffix" is one under which Internet users can directly register names. Some examples of public suffixes are ".com", ".co.uk" and "pvt.k12.wy.us".*[11]

When a domain is not a public suffix it is generally under the control of a single entity and the maliciousness of subdomains are in most cases closely related to the maliciousness of the top domain. For public suffixes this is not the case.

It is generally safe to assume that the public suffixes themself are not malicious. It is therefore desirable to whitelist public suffixes, but the whitelisting of public suffixes should not influence the maliciousness of their sub domains. The underlying reason for this is that even though the service provider is legitimate, their service can be used for malicious purposes. For example, the dynamic

[11] http://publicsuffix.org/

DNS provider dyndns.org is legitimate but they might unknowingly sell the domain evil.dyndns.org to cybercriminals.

Unfortunately, there is no perfect method to determine if a domain is a public suffix. There are however some best effort community initiatives. The most promising is the "public suffix list" initiative from Mozilla[12]. Another interesting initiative is the community list of dynamic domain name providers started by Malware Domain blocklist[13]

Because it is not possible to determine the public suffix with certainty, it is desirable that the whitelisting algorithm is conservative when removing domains from the black signature set because the super domain is whitelisted. Normally a domain is removed from the blacklist if the domain is also in the whitelist and the whitelist score is higher than the blacklist score. To add an extra safety when removing subdomains a requirement is added that the whitelist score must be 40% higher before a sub domain is removed from the blacklist.

When the whitelisted domain is not a public suffix the decision to remove a domain from the black signature set can there for be expressed mathematically as:

$$black_{risk\_score} * 1.4^{domain\_level} > white_{risk\_score}$$

The following 3 examples illustrate the algorithm:

- Example 1: The domain *evil.com* is blacklisted with a risk score of 90 and *evil.com* is whitelisted with a score of 80. In this example *evil.com* will not be removed from the blacklist because $90 * 1.4^0 > 80$
- Example 2: The domain *ns.evil.com* is blacklisted with a risk score of 70 and *evil.com* is whitelisted with a score of 80. In this example *ns.evil.com* will not be removed from the blacklist because $70 * 1.4^1 > 80$
- Example 3: The domain *ns1.ns.evil.com* is blacklisted with a risk score of 60 and *evil.com* is whitelisted with a score of 80. In this example *ns1.ns.evil.com* will not be removed from the blacklist because $60 * 1.4^2 > 80$

## 3.3 Matching engine

The matching engine takes as input the signatures and the log data and generates alarms.

The number of blacklists signatures are in the range 100 000 to 1 000 000. The matching engine must be able to search through many gigabytes of log data quickly. This requires that the matching engine uses a search algorithm that has good asymptotic run time properties with regard to size of the signature set and the size of the search text. A good candidate algorithm is the Aho-Corasick

[12] http://www.malware-domains.com/files/dynamic_dns.zip
[13] http://mxr.mozilla.org/mozilla-central/source/netwerk/dns/effective_tld_names.dat?raw=1

multi string matching algorithm(Aho & Corasick, 1975). The algorithm has a runtime that is linear to the length of the patterns plus the length of the searched text plus the number of output matches.

Using a fixed string search algorithm represents some problems. For example searching for the IP-address 10.0.0.1 will also match on the IP-address 110.0.0.1. The same problem with over matching can be found when searching for domains. For example searching for the domain evil.com will also match on the domain notevil.com. Another problem is that it is not possible to search for IP-address networks. For example, 192.168.1.0/24 will not give a match on 192.168.1.1.

To work around this problem, the matching engine will have to use a preprocessing and post processing step to prepare the signature and verify the matches. For example, the signature "192.168.1.0/24" needs to be changed to "192.168.1." in the preprocessor step and if the signature matches the post processing step must check that the match actually is an IP-address in the correct range.

Currently support for Domains, IPv4-adresses, IPv4-ranges and IPv4-CIDRs has been implemented. But it is easy to extend this to other types of signatures as long as a pre-processing function can generate a fixed string guard for the signature. The requirement of the fixed string guard is that it cannot produce false negatives and for performance reasons the number of false positives should be as low as possible.

Below is a simple example demonstrating how the matching engine works. When the matching engine is loaded with the following signatures:

**Table 4 Example signature set**

| |
|---|
| 92.122.190.0/24<br>148.123.13.68<br>n3g.akamai.net |

And feed with the following log data:

**Table 5 Example log data**

| |
|---|
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n1g.akamai.net. 1138 A 92.122.190.85<br>2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n2g.akamai.net. 2241 A 148.123.13.63<br>2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n3g.akamai.net. 644 A 148.123.13.68 |

It will out put the following two alarms

**Table 6 Example alarmdata**

| Sigs | Data |
|---|---|
| 92.122.190.0/24 | 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n1g.akamai.net. 1138 A 92.122.190.85 |
| 148.123.13.68 | 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n3g.akamai.net. 644 A |

| | |
|---|---|
| n3g.akamai.net | 148.123.13.68 |

Each alarm contains the signatures that matched the line, the position they matched and the raw log line:

## 3.4 Quantifying victim risk

When the botnet detection software is in use, a matching engine will use the signature set and match it against activity logs. If an event matches any of the signatures, it will generate an alarm. The previously described relational model for signatures can be extended to also include the newly generated alarms.

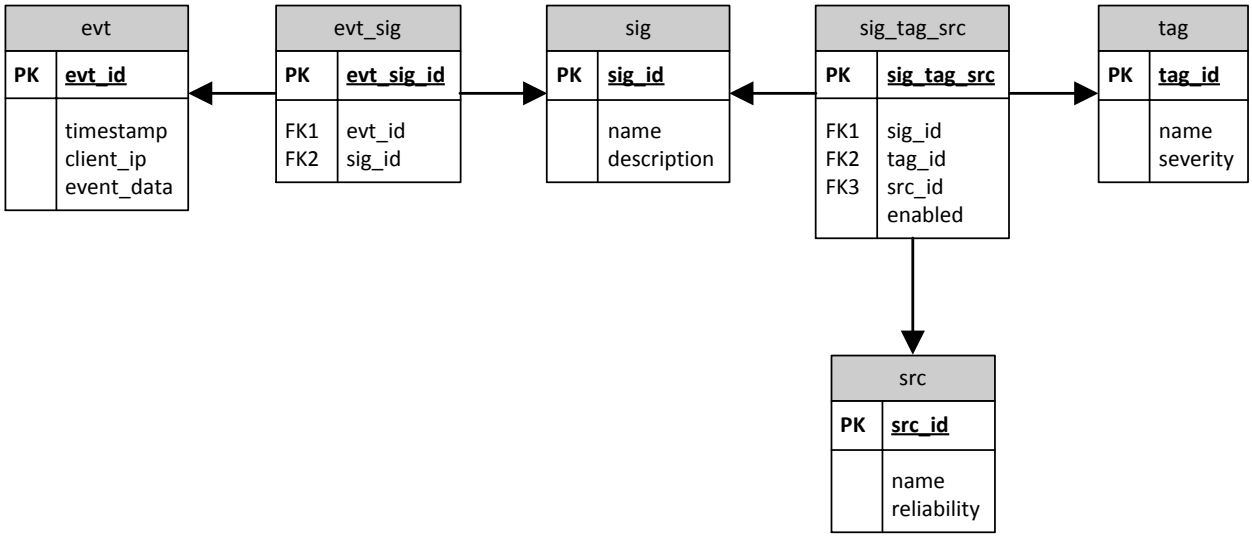



Figure 16: Extended relational model, including alarms

This model allows for each alarm to be linked to multiple signatures. The total risk for a given alarm can then be found by aggregating the risk from each signature. Having a good event risk score is very useful but in terms of operational incident handling, dealing with events is problematic. The main reason is the speed at which things happen on a large network and the amount of alarms that can be generated. What most people actually want to know is if any of the computers in their network are compromised or not. To be able to answer this question there is a need for aggregating the risk score for individual alarms into a combined risk score for each host on the network

There are certain challenges when designing a host risk score. Computers in a network are infected and clean up all the time, calculated risk score must therefore take this into consideration. One way to deal with this is to calculate the risk score for fixed time intervals. An interval of one hour or one day seems like a reasonable starting point. If the interval is too long, a computer that was infected but is now cleaned up might appear to be affected by the algorithm. If the interval is too short there can also be problems, an infected computer might seem to be clean in some of the intervals just because there was no activity even though the computer is still infected. It is therefore desirable to have as short intervals as possible, but still avoiding the problems related to a computer starting to appear clean while it's not. Applying a smoothing algorithm would probably improve the estimated risk score, and avoid fluxing risk score due to inactivity.

Within a given timeframe, a single computer might trigger the same signature many times and it might trigger many different signatures. The challenge is to finding an algorithm that can combine all the alarms in such a way that the computer risk score represent the likelihood that the computer is compromised.

Sometimes user activity can generate network traffic that triggers alarms without the computer really being infected. This is considered a false positive. There are many underlying reason for false positives. One typical issue is that signature is too broad and matches on unrelated activity. This can for example happen if a signature matches on random data from an encrypted session. Another problem is that the signature is no longer correct, it matches on what it's intended to match on but it is not malicious. This can for example happen if an IP-address once was used for malicious activity but is now cleaned up.

In most networks a lot of the diversity in network traffic comes from encrypted protocols or user activity like web surfing and mail. A lot of this traffic will only be seen once. If a signature is too broad and by accident matches on some of this traffic, it will normally only trigger once or a few times. From this we can see that a lot of the false positives that happens because the signature is not precise enough comes from encrypted traffic or user activity, since this traffic is less repetitive than other types of traffic this type of false positives will normally trigger very few false positives. This means that the likelihood that a signature triggers false positives decreases with the number of alarms it has triggered from a specific host. An example of this is if a user by accident enters a bad URL in the web browser and visits a page that is associated with a botnet command and control server. This will trigger a botnet alarm, but the alarm will normally only be triggered once. In contrast, if a computer is recruited into a botnet it is normal for the computer to be in regular contact with the command and control servers. From this we can see that as the number of times a signature has triggered an alarm within a given time period the likelihood that the computer is compromised increases

Even though the likelihood that a computer is really infected increases with the amount of alarms an individual signature has triggered, there is always risk that the signature is wrong and the alarms are false positives. For example in some cases a signature can simply be wrong and trigger on normal legitimate traffic. In such cases, the signature can create enormous amounts of false alarms. This will very often happen for signatures from sources of low reliability. For example if an unreliable source adds “facebook.com” to signature list because of a “facebook virus” this will no doubt generate many false positives. This illustrated the point that we do not want signatures with low risk score to dominate the client risk score even if they trigger many alarms.

From this, it seems reasonable that when calculating the risk score from a computer the contribution to the total risk from a single signature should increase with the number of alarms it has generated, but that the contributed from each new alarm should decrease in such a fashion that the number never dominates the signature risk score. This seems to indicate that the client risk score contribution function as a function of the number events should converge on a number. A reasonable estimate might be that the risk contribution from the number of alarms should be no

more than a factor of 3. A function that has the desired properties that when n goes from 1->∞ the risk contribution function goes from 1->4 would be as following:

$$\frac{4}{1+\frac{3}{number\_of\_alarms}}$$

Given the arguments above the client risk score contribution from one signature can then be expressed like this:

$$client\_risk\_score = \frac{4}{1+\frac{3}{number\_of\_alarms}} * signature\_risk\_score$$

For combining the risk score contributions from each signature into a total client risk score it seems reasonable to use the same RSS algorithm that was used to calculate the total risk score for signatures. To formalize the expression for total client risk score will need some additional terminology.

A set of clients are described with a capital C, a particular client with a letter case c. The risk score associated with the computer is given by the term $c_{score}$. The term $n_r$ is used to denote the number of times the signature has triggered an alarm for the given client.

The total client risk score can then be expressed as:

$$c_{score} = \sqrt{\sum_{uniq\ R_c} \left( \frac{4}{1+\frac{3}{n_r}} * r_{score} \right)^2}$$

# 4 Data collection

## 4.1 Log data

The Log data for the experiment was gathered over period of 1 month from 1 April 2012 to 1 May 2012 from an organization in Norway.

Two types of logdata were recorded, passive DNS and Argus Netflow. The logdata was gathered from a passive IDS sensor. The IDS sensor was listening on a monitor port on a switch that received all traffic going between the organization and the Internet.

In the period 41 gigabytes of passive dnslog data was recorded. That amounts to a total of 458 116 583 log entries. For Argus Netflow 109 gigabytes of log data was recorded. That amounts to a total of 1 016211308 log entries.

### 4.1.1 Example Log data

The network activity was recorded in the form of line based log data. Each line of log data represents a distinct event on the network. For DNS the following attributes were collected: timestamp, client IP, cliet port, server IP, server port, dns query id, and the dns response. Each DNS RR record is stored on a separate line.

**Table 7: Example DNS log**

| |
|---|
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n1g.akamai.net. 1138 A 92.122.190.85 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n2g.akamai.net. 2241 A 148.123.13.63 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n3g.akamai.net. 644 A 148.123.13.68 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n4g.akamai.net. 1140 A 148.123.13.69 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n5g.akamai.net. 2979 A 148.123.13.79 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n6g.akamai.net. 705 A 148.123.13.77 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n7g.akamai.net. 1138 A 148.123.13.63 |
| 2011-09-27 20:46:34.246279 193.213.112.4 53 192.168.1.7 65105 n8g.akamai.net. 3087 A 148.123.13.76 |
| 2011-09-27 20:46:34.247868 193.213.112.4 53 192.168.1.7 58944 a4.sphotos.ak.fbcdn.net. 4458 CNAME a4.sphotos.ak.fbcdn.net.edgesuite.net. |
| 2011-09-27 20:46:34.247868 193.213.112.4 53 192.168.1.7 58944 a4.sphotos.ak.fbcdn.net.edgesuite.net. 17358 C |

For netflow there are many different log formats. For this paper the tool Argus was used to generate the log data. Each network session is stored on a single line. The following attributes was stored for each network session: start time, end time, client IP, client port, server IP, server port, client bytes, server bytes.

**Table 8: Example Netflow log**

| |
|---|
| 2011-09-27 14:33:39\|2011-09-27 14:33:57\|8\|17\|192.168.1.7\|52327\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |
| 2011-09-27 14:33:25\|2011-09-27 14:33:38\|8\|17\|192.168.1.7\|59403\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |
| 2011-09-27 14:33:10\|2011-09-27 14:33:15\|8\|17\|192.168.1.7\|62742\|193.213.112.4\|53\|2\|1\|82\|41\|0\|0\|0\|0 |
| 2011-09-27 14:29:28\|2011-09-27 14:29:58\|8\|17\|192.168.1.7\|52050\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |
| 2011-09-27 14:29:42\|2011-09-27 14:29:50\|8\|17\|192.168.1.7\|63813\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |
| 2011-09-27 14:29:13\|2011-09-27 14:29:40\|8\|17\|192.168.1.7\|63135\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |
| 2011-09-27 14:27:04\|2011-09-27 14:27:22\|8\|17\|192.168.1.7\|64691\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |
| 2011-09-27 14:26:48\|2011-09-27 14:26:56\|8\|17\|192.168.1.7\|49489\|193.213.112.4\|53\|3\|1\|123\|41\|0\|0\|0\|0 |

```
2011-09-27 14:26:35|2011-09-27 14:26:55|8|17|192.168.1.7|52014|193.213.112.4|53|3|1|123|41|0|0|0|0
2011-09-27 14:25:44|2011-09-27 14:25:59|8|17|192.168.1.7|65032|193.213.112.4|53|3|1|123|41|0|0|0|0
```

## 4.2 Signatures

The signatures were obtained from publicly available blacklist. The signatures consist of domains, IP-addresses and IP-networks in CDIR format.

Table 9 Signature overview

| Signature list | Signature count |
|---|---|
| Zeus Tracker | 642 |
| Spyeye Tracker | 321 |
| Spamhaus DROP | 421 |
| Sedb | 100988 |
| Malewaredomainlist | 60643 |
| Malwaredomains | 15733 |
| *Total* | *178746* |

Summing the signatures from each source gives a total count of 178 746 signatures. Because some signatures are found in more than one signature list the total number of unique signatures in the signature database is less than the total. The signature database contained 175 935 signatures. 2811 of signatures are therefor reported in more than one signature list

### 4.2.1 Zeus Tracker

The Zeus tracker blacklist is hosted at zeustracker.abuse.ch. Zeus tracker provides IP and domain information for command and control servers using the Zeus crimeware kit. The ZeuS crimeware kit is designed to steal credentials from various online services like social networks, online banking accounts, ftp accounts, email accounts and other.

The domains and IP-addresses provided by Zeus tracker is very reliable and creates few false positives. Since the methods used to generate the list is not publicly known it is difficult to assess how complete the list is.

### 4.2.2 Spyeye Tracker

The Spyeye Tracker is hosted at spyeyetracker.abuse.ch. Spyeye is a crimeware kit is similar to Zeus. Because the list is created by the same team that creates spyeye tracker it is reasonable to assume that they are generated using the same mechanism

### 4.2.3 Spamhaus DROP

The spamhaus drop list is a list of netblocks that are considered to be hijacked or stolen and entirely under the control of criminals or professional spammers. Any traffic to these net blocks is suspicious. Because this list consists of net blocks these signatures cannot be directly used in a string based search algorithm

Table 10: Example netbloks

```
2.56.0.0/14 ; SBL102988
14.192.0.0/19 ; SBL123577
14.192.48.0/21 ; SBL131019
14.192.56.0/22 ; SBL131020
31.11.43.0/24 ; SBL113323
31.135.0.0/21 ; SBL137007
```

### 4.2.4 SpyEye Database (Sedb)

This is a set of domains and IP-addresses related to SpyEye. The list is created by the same team that creates the SpyEye Tracker

### 4.2.5 Malewaredomainlist

The list contains a large amount of domains and IP-addresses related to malware. Some of the signatures are bad and create many false positives. The list provides some context information making it possible to tag some of the signature with more specific tags like for example botnet, exploit, spyeye, zeuz etc.

### 4.2.6 Malwaredomains

A large list of domains very similar to malwaredomainlist

# 5 Results and Discussion

In this chapter the proposed system described in *Chapter 3: Methodology* is tested using the data and signatures described in *Chapter 4: Data Collection*.

The first section of this chapter describes how the log files are searched using a simple string search and the result from the search is analyzed. In the two next sections, features from the system described in chapter 3 are introduces one by one. The same log data is searched and the results are analyzed. The results are then compared to the result in the previous sections to evaluate if the newly introduced features reduce the number of false positives. The main goal of the 3 first sections is to answer *RQ1: Will the system be able to automatically deal with signatures that generate large amount of false positives?*

Section 4 looks at the aggregated victim scores and tries to answer *RQ2: Will the system simplify the process of detecting compromised host in the network compared to manual inspecting each alarm?*

The last section reviews the performance of the proposed system.

## 5.1 Signature search

In this section the log data is searched using a simple string search algorithm. For this task the Linux command tool grep is used.

The following command was used:

```
grep --color=always -F -f siglist.txt logfile.txt
```

When grep is given the '-F' flag it does a fixed string search using the Aho–Corasick string matching algorithm enabling grep to search the logfile with a large signature set very efficiently. The signatures are listed in the siglist.txt file, one signature per line. The '–color=always' flag makes sure that the part of the log that matches a signature are colored making it easier to analyze the output. Below is a simple example illustrating how grep works:

**Table 11 Example siglist.txt**

```
evil.com
superevil.com
```

**Table 12 Example logfile.txt:**

```
2011-09-27 14:13:30 192.168.1.1 53 192.168.1.7 evil.com 454 A 122.224.5.45
2011-09-27 14:13:30 192.168.1.1 53 192.168.1.7 62694 superevil.com 454 NS ns1.evi.com.
2011-09-27 14:13:30 192.168.1.1 53 192.168.1.7 62694 hdcmct.com. 454 NS f1g1ns2.dnspod.net.
2011-09-27 14:13:30 192.168.1.1 53 192.168.1.7 62694 hdcmct.com. 454 NS f1g1ns2.dnspod.net.
```

The output of running grep with the parameters given above on the sample files would be:

Table 13 Grep output

```
2011-09-27 14:13:30 192.168.1.1 53 192.168.1.7 evil.com 454 A 122.224.5.45
2011-09-27 14:13:30 192.168.1.1 53 192.168.1.7 62694 superevil.com 454 NS
ns1.evil.com.
```

We can see that that the output from the grep command only contains the lines that has a substring listed in siglist.txt. The original logfile has 4 log lines while the output of grep only has 2 lines. In the example above evil.com would also match on superevil.com, but this is not a problem because the variant of aho-Corasick that is used returns only nonoverlapping matches and it chooses the longest if there are conflicting matches. For this reason the whole domain superevil.com will always be matched in the above example.

To get a baseline of the number of alarms the different signature lists create the grep command described above was used to test on log data from one day. The command was run on log data from 30th April 2012. The log file was 2.3 GB and contained 24.609.527 log entries. The table below describes the number of alarms generated from each signature list.

Table 14 Alarm baseline

| Signature list | Signature Count | Alarms count |
|---|---|---|
| Zeus Tracker | 648 | 0 |
| Spyeye Tracker | 316 | 0 |
| Spamhaus DROP | 421 | 0 |
| Sedb | 100147 | 659646 |
| Malewaredomainlist | 60404 | 63729 |
| Malwaredomains | 15946 | 786 |
| Total | *177882* | 724161 |
| All | 174910 | 724107 |

We can see that for some of the lists the number of alarms is extremely high and it clear only from looking at the number of alarms that most of them are probably false positives. The reliable blacklist Zeus and Spyeye Tracker reported zero alarms; this might be an indication that the network is relatively clean. The zero alarms from the Spamhaus DROP list are simply because the signatures are in the CDIR format and the string search algorithm does not understand this signature format. For example the signature 192.168.1.0/24 will not give a match on 192.168.1.2 even though it is an IP-address within the CDIRs range.

The extremely large number of alarms from the Sedb and malwaredomainlist makes it very hard to work with the alarms directly. Quickly scrolling through some of the signatures indicates that most of the alarms are generated by signatures that clearly are wrong.

For example are www.geocities.com, sites.google.com, www.l.google.com, s3.amazonaws.com, ad.yieldmanager.com listed as malicious. The sites www.geocities.com, sites.google.com and s3.amazonaws.com are popular legitimate services but because they offer user to host their own content they are sometimes used to host malicious content and this can result in the pages being

added to the blacklist. The domain www.l.google.com is a cname for www.google.com. Some malware automatically connects to www.google.com to test if they have internet access. Because of this some automated malware analysis software might have labeled the domain as malicious.

## 5.2 Signature search with automated whitelisting

The simple signature search in the previous section illustrated that performing a signature search directly without removing some of the bad signatures will generated too many false positives to be useful. In this section, we apply the whitelisting algorithm described in section 6.2 to remove signatures that should not be in the signature set. The table below illustrates the number of signatures that were removed from each signature list by the whitelisting algorithm

Table 15 Results from whitelisting

| Signature list | Signature count | whitelisted | After whitelisting | Reduction |
|---|---|---|---|---|
| Zeus Tracker | 648 | 0 | 648 | 0,00 % |
| Spyeye Tracker | 316 | 0 | 316 | 0,00 % |
| Spamhaus DROP | 421 | 0 | 421 | 0,00 % |
| Sedb | 100147 | 238 | 99909 | 0,24 % |
| Malewaredomainlist | 60404 | 209 | 60195 | 0,35 % |
| Malwaredomains | 15946 | 131 | 15815 | 0,82 % |
| *Total* | *177882* | 578 | 177304 | 0,32 % |
| *All* | 174910 | 573 | 174337 | 0,33 % |

We can see from the table above that the total number signatures that where whitelisted is quite small. Less than 1% for all signature sets. This result is not very surprising. It was expected that only a small number of signatures was responsible for most of the false positives in the previous section. In total 988 domains were whitelisted.

With such any whitelisting there is always a risk of whitelisting malicious signatures and in the process possibly creating false negatives, but since only 573 of 174910 signatures where whitelisted the risk for false negatives is quite small. It would also be possible to go back and manually re-enable signatures that was wrongly whitelisted

Rerunning the simple signature search from section 7.1 with the refined signature set gave the following result:

Table 16 Results from whitelisting

| Signature list | Alarms | Alarms after whitelisting | Reduction |
|---|---|---|---|
| Zeus Tracker | 0 | 0 | 0,00 % |
| Spyeye Tracker | 0 | 0 | 0,00 % |
| Spamhaus DROP | 0 | 0 | 0,00 % |
| Sedb | 659646 | *426* | 99,94 % |
| *Malewaredomainlist* | 63729 | *7558* | 88,14 % |
| Malwaredomains | 786 | 416 | 47,07 % |
| Total | 724161 | 8400 | 98,84 % |
| All | 724107 | 8393 | 98,84 % |

We see that for the Sedb *and Malewaredomainlist* the number of alarms was reduced by 88% and 99%. Most of these are a generated by a few signatures that triggered on popular legitimate domains. The reduction of false positives by 98% in total is a dramatic improvement and illustrates that a good whitelisting algorithm is very important when dealing with large signature sets. It also indicates that the whitelisting approach suggested here is better than the whitelisting method currently in operational use by the people generating these lists.

The total number of alarms is still over 8000 for only one day of DNS log data. That is still too many alarms to deal with manually and further improvements are required to easily be able to detect compromised computes.

From manually inspecting the alarms, it is clear that some of the false positives are generated because of the limitations of the fixed string search algorithm. For example, the signature for the domain *collective-media.ne* also triggers on the domain *a.collective-media.net.edgekey.net* and this creates false positives.

.

## 5.3 Improved signature matching

The problem with the fixed string matching used so far is that it can generate a lot of false positives because of over matching. For example if the domain ogle.co is used as a command and control server and is added to any of the signature list, this will generate a lot of false positives because "ogle.co" is a substring of "google.com". A simple fixedstring search in the log files is not accurate enough to avoid false positives from overmatching.

Below is the result from rerunning the search described in section 7.2, but replacing grep with the matching engine described in section 6.3

Table 17 Results from usning Matchin engine

| Signature list | Alarms after whitelisting | Alarms after using MatchingEngine | Reduction |
|---|---|---|---|
| Zeus Tracker | 0 | 0 | 0,00 % |
| Spyeye Tracker | 0 | 0 | 0,00 % |
| Spamhaus DROP | 0 | 0 | 0,00 % |
| Sedb | *426* | *57* | 86,62 % |
| *Malewaredomainlist* | *7558* | 6797 | 10,07 % |
| Malwaredomains | 416 | *103* | 75,24 % |
| Total | 8400 | 6957 | 17,18 % |
| All | 8393 | 6955 | 17,13 % |

The result illustrates the practical difference of changing the matching algorithm from the fixed string algorithm in grep to a more sophisticated matching engine that understands domains, IP-addresses and IP-ranges. The improvement is not quite as dramatic as the first whitelisting but removing 17% of the alarms or in total removing 1438 false positives is still very helpful.

## 5.4 Aggregating alarms by victim

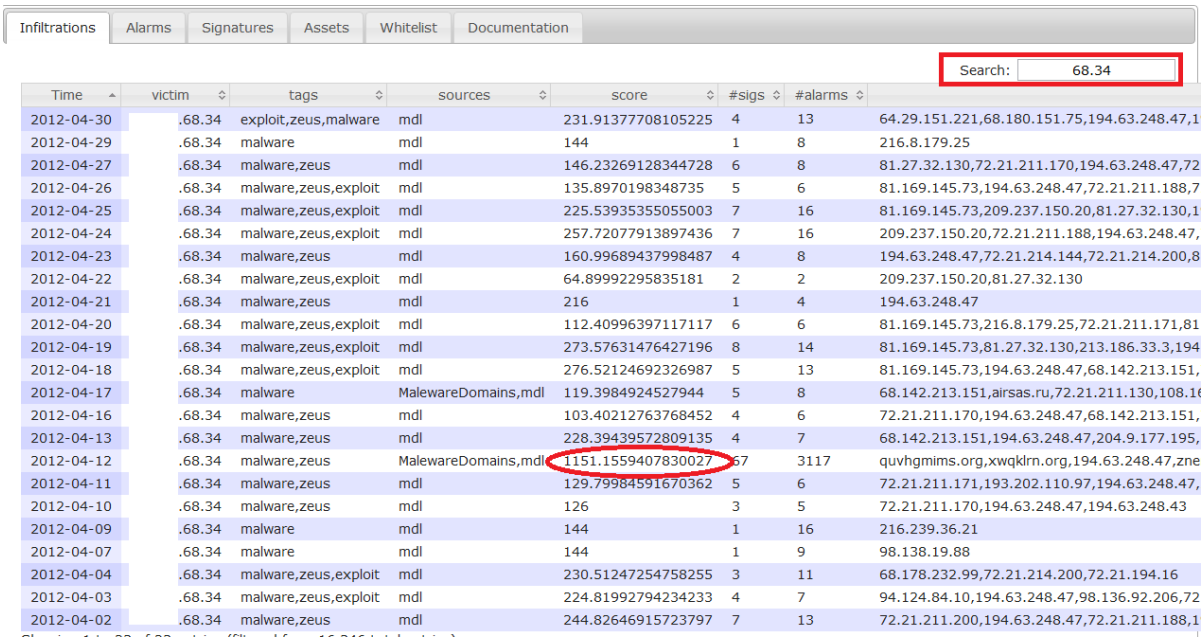
To test if it was possible to detect compromised hosts based on the aggregated host risk scores the signatures set described in previous section was used to search an entire month of DNS log data. In total 122 638 alarms was generated.

To be able to more easily view the alarms and aggregated victim scores a simple web frontend was built. Figure 17 below lists the hosts with the highest risk score per day. Since this is real-life log data part of the victim IP-address is removed for privacy concerns.

| Time | victim | tags | sources | score | #sigs | #alarms |
|---|---|---|---|---|---|---|
| 2012-04-12 | .68.34 | malware,zeus | MalewareDomains,mdl | 1151.1559407830027 | 67 | 3117 |
| 2012-04-12 | .84.178 | malware | MalewareDomains | 1065.5026982603094 | 63 | 937 |
| 2012-04-20 | .23.197 | exploit,zeus,malware | mdl | 455.36798306424663 | 17 | 39 |
| 2012-04-17 | .156.5 | malware,zeus,exploit | mdl | 445.29540756670735 | 7 | 38 |
| 2012-04-18 | .152.194 | malware,zeus | mdl | 443.8378082137663 | 9 | 49 |
| 2012-04-17 | .142.238 | malware,zeus,exploit | mdl | 438.32864382789313 | 15 | 48 |
| 2012-04-11 | .80.170 | malware,zeus,exploit | mdl | 428.98951036126743 | 11 | 58 |
| 2012-04-11 | .139.8 | malware,zeus,exploit,botnet | mdl | 428.1495065978705 | 8 | 637 |
| 2012-04-10 | .80.170 | malware,zeus | mdl | 425.9577443831724 | 9 | 46 |
| 2012-04-30 | .63.130 | malware,zeus | mdl | 411.25174771665104 | 12 | 62 |
| 2012-04-20 | .161.35 | malware,zeus | mdl | 410.8576395784798 | 9 | 68 |
| 2012-04-13 | .42.36 | malware,zeus | mdl | 410.8576395784798 | 9 | 107 |
| 2012-04-16 | .152.194 | malware | MalewareDomains,mdl | 408.8814008976197 | 9 | 49 |
| 2012-04-30 | .155.175 | malware | mdl | 407.29350596345137 | 8 | 60 |
| 2012-04-30 | .42.36 | malware | mdl | 407.29350596345137 | 8 | 102 |
| 2012-04-29 | .42.36 | malware | mdl | 407.29350596345137 | 8 | 94 |
| 2012-04-29 | .161.35 | malware | mdl | 407.29350596345137 | 8 | 101 |
| 2012-04-29 | .155.175 | malware | mdl | 407.29350596345137 | 8 | 58 |
| 2012-04-29 | .48.136 | malware | mdl | 407.29350596345137 | 8 | 51 |
| 2012-04-29 | .48.140 | malware | mdl | 407.29350596345137 | 8 | 56 |
| 2012-04-28 | .48.140 | malware | mdl | 407.29350596345137 | 8 | 86 |
| 2012-04-28 | .48.136 | malware | mdl | 407.29350596345137 | 8 | 64 |
| 2012-04-28 | .42.36 | malware | mdl | 407.29350596345137 | 8 | 93 |
| 2012-04-28 | .161.35 | malware | mdl | 407.29350596345137 | 8 | 112 |
| 2012-04-28 | .155.175 | malware | mdl | 407.29350596345137 | 8 | 62 |
| 2012-04-27 | .161.35 | malware | mdl | 407.29350596345137 | 8 | 101 |
| 2012-04-27 | .48.140 | malware | mdl | 407.29350596345137 | 8 | 63 |
| 2012-04-27 | .42.36 | malware | mdl | 407.29350596345137 | 8 | 97 |
| 2012-04-26 | .42.36 | malware | mdl | 407.29350596345137 | 8 | 105 |

**Figure 17 Network host with highest risk score per day**

The IP-address with the highest score is "68.34" and it had a total risk score of 1151 on the 12.04.2012. On the that particular day "68.24" was in contact with 67 different network resources known to be bad and triggered in total 3117 alarms. To get more details about the host the table can be filtered by the IP-address and sorted by day. Figure 18 below illustrates this.



Infiltrations | Alarms | Signatures | Assets | Whitelist | Documentation

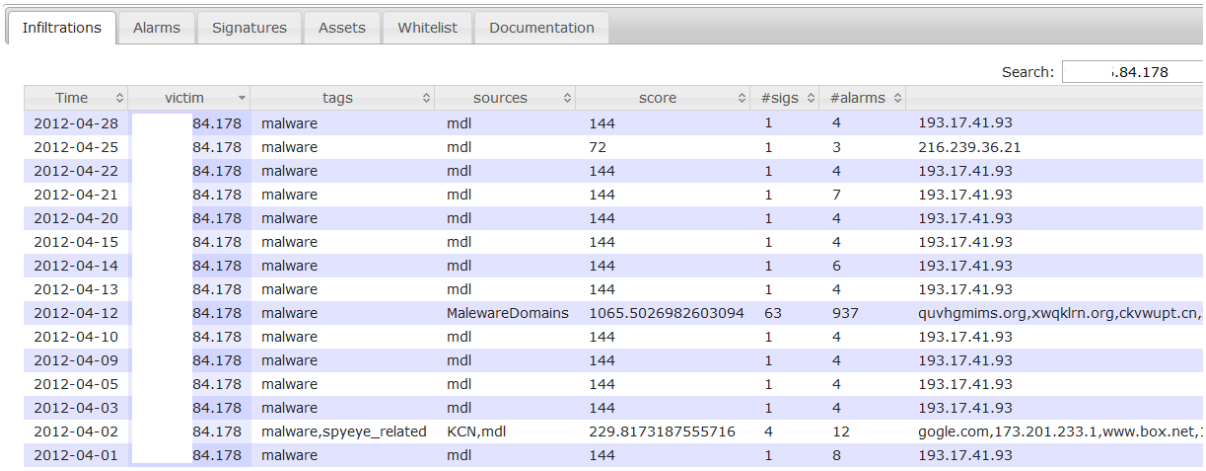
Search: 68.34

| Time | victim | tags | sources | score | #sigs | #alarms | |
|---|---|---|---|---|---|---|---|
| 2012-04-30 | .68.34 | exploit,zeus,malware | mdl | 231.91377708105225 | 4 | 13 | 64.29.151.221,68.180.151.75,194.63.248.47,1 |
| 2012-04-29 | .68.34 | malware | mdl | 144 | 1 | 8 | 216.8.179.25 |
| 2012-04-27 | .68.34 | malware,zeus | mdl | 146.23269128344728 | 6 | 8 | 81.27.32.130,72.21.211.170,194.63.248.47,72 |
| 2012-04-26 | .68.34 | malware,zeus,exploit | mdl | 135.8970198348735 | 5 | 6 | 81.169.145.73,194.63.248.47,72.21.211.188,7 |
| 2012-04-25 | .68.34 | malware,zeus,exploit | mdl | 225.53935355055003 | 7 | 16 | 81.169.145.73,209.237.150.20,81.27.32.130,1 |
| 2012-04-24 | .68.34 | malware,zeus,exploit | mdl | 257.72077913897436 | 7 | 16 | 209.237.150.20,72.21.211.188,194.63.248.47, |
| 2012-04-23 | .68.34 | malware,zeus | mdl | 160.99689437998487 | 4 | 8 | 194.63.248.47,72.21.214.144,72.21.214.200,8 |
| 2012-04-22 | .68.34 | malware,zeus,exploit | mdl | 64.89992295835181 | 2 | 2 | 209.237.150.20,81.27.32.130 |
| 2012-04-21 | .68.34 | malware,zeus | mdl | 216 | 1 | 4 | 194.63.248.47 |
| 2012-04-20 | .68.34 | malware,zeus,exploit | mdl | 112.40996397117117 | 6 | 6 | 81.169.145.73,216.8.179.25,72.21.211.171,81 |
| 2012-04-19 | .68.34 | malware,zeus,exploit | mdl | 273.57631476427196 | 8 | 14 | 81.169.145.73,81.27.32.130,213.186.33.3,194 |
| 2012-04-18 | .68.34 | malware,zeus,exploit | mdl | 276.52124692326987 | 5 | 13 | 81.169.145.73,194.63.248.47,68.142.213.151, |
| 2012-04-17 | .68.34 | malware | MalewareDomains,mdl | 119.3984924527944 | 5 | 8 | 68.142.213.151,airsas.ru,72.21.211.130,108.1 |
| 2012-04-16 | .68.34 | malware,zeus | mdl | 103.40212763768452 | 4 | 6 | 72.21.211.170,194.63.248.47,68.142.213.151, |
| 2012-04-13 | .68.34 | malware,zeus | mdl | 228.39439572809135 | 4 | 7 | 68.142.213.151,194.63.248.47,204.9.177.195, |
| 2012-04-12 | .68.34 | malware,zeus | MalewareDomains,mdl | 1151.1559407830027 | 67 | 3117 | quvhgmims.org,xwqklrn.org,194.63.248.47,zne |
| 2012-04-11 | .68.34 | malware,zeus | mdl | 129.79984591670362 | 5 | 6 | 72.21.211.171,193.202.110.97,194.63.248.47, |
| 2012-04-10 | .68.34 | malware,zeus | mdl | 126 | 3 | 5 | 72.21.211.170,194.63.248.47,194.63.248.43 |
| 2012-04-09 | .68.34 | malware | mdl | 144 | 1 | 16 | 216.239.36.21 |
| 2012-04-07 | .68.34 | malware | mdl | 144 | 1 | 9 | 98.138.19.88 |
| 2012-04-04 | .68.34 | malware,zeus,exploit | mdl | 230.51247254758255 | 3 | 11 | 68.178.232.99,72.21.214.200,72.21.194.16 |
| 2012-04-03 | .68.34 | malware,zeus,exploit | mdl | 224.81992794234233 | 4 | 7 | 94.124.84.10,194.63.248.47,98.136.92.206,72 |
| 2012-04-02 | .68.34 | malware,zeus | mdl | 244.82646915723797 | 7 | 13 | 72.21.211.200,194.63.248.47,72.21.211.188,1 |

**Figure 18 Compromised host table filtered by "68.34" and sorted by day**

Only from looking at this data it is quite clear that "68.34" was exposed to something malicious on 4/12. A simple Google search on some of the Domains and IP-addresses that triggered the alarms reveals that some of the domains are associated with the Conficker bot. It is therefore very likely that the host in question is infected with Conficker.

Checking out the next victim "84.178" from figure 17 reveals that he has also been contacting the same domains and is therefore also most likely infected with Conficker. Figure 19 below list the victim score of "84.178" by day.



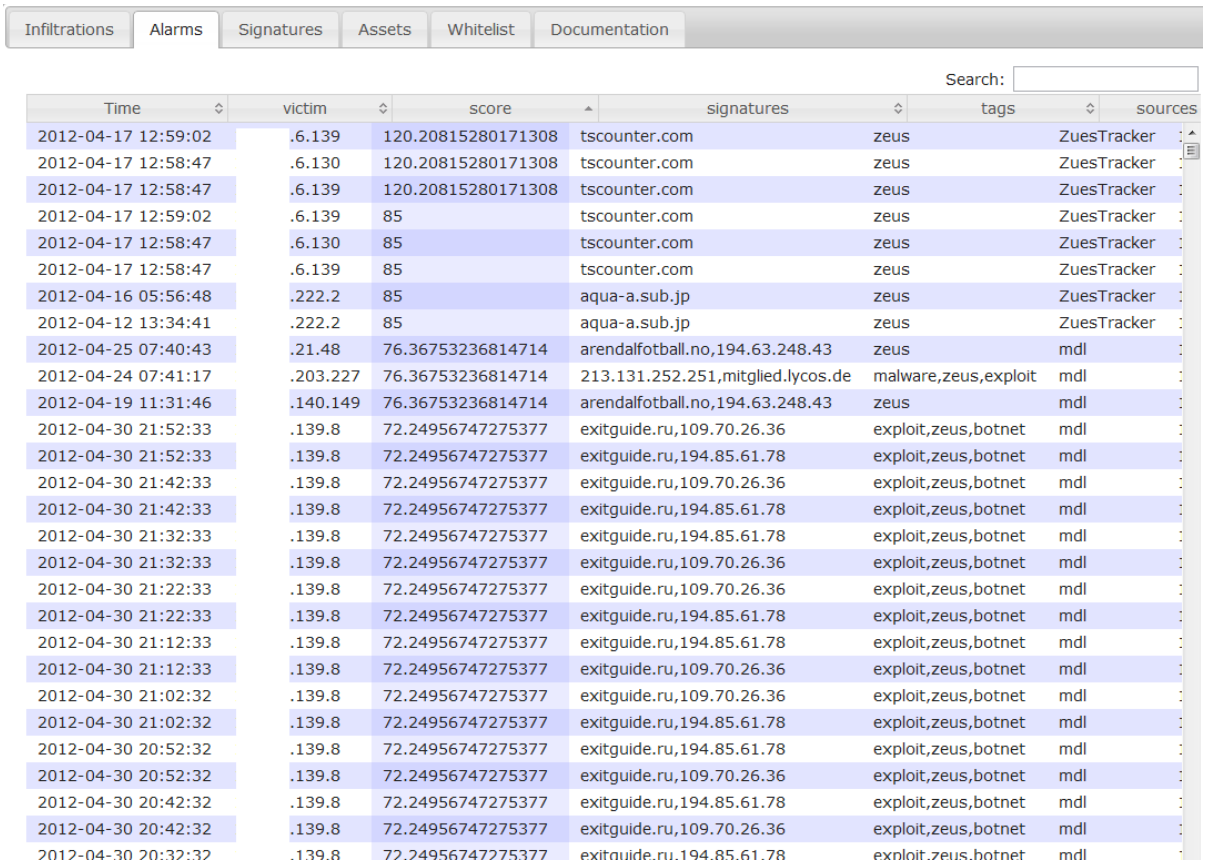
Infiltrations | Alarms | Signatures | Assets | Whitelist | Documentation

Search: .84.178

| Time | victim | tags | sources | score | #sigs | #alarms | |
|---|---|---|---|---|---|---|---|
| 2012-04-28 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-25 | 84.178 | malware | mdl | 72 | 1 | 3 | 216.239.36.21 |
| 2012-04-22 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-21 | 84.178 | malware | mdl | 144 | 1 | 7 | 193.17.41.93 |
| 2012-04-20 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-15 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-14 | 84.178 | malware | mdl | 144 | 1 | 6 | 193.17.41.93 |
| 2012-04-13 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-12 | 84.178 | malware | MalewareDomains | 1065.5026982603094 | 63 | 937 | quvhgmims.org,xwqklrn.org,ckvwupt.cn, |
| 2012-04-10 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-09 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-05 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-03 | 84.178 | malware | mdl | 144 | 1 | 4 | 193.17.41.93 |
| 2012-04-02 | 84.178 | malware,spyeye_related | KCN,mdl | 229.8173187555716 | 4 | 12 | gogle.com,173.201.233.1,www.box.net, |
| 2012-04-01 | 84.178 | malware | mdl | 144 | 1 | 8 | 193.17.41.93 |

**Figure 19 Compromised host table filtered by "84.|78" and sorted by day**

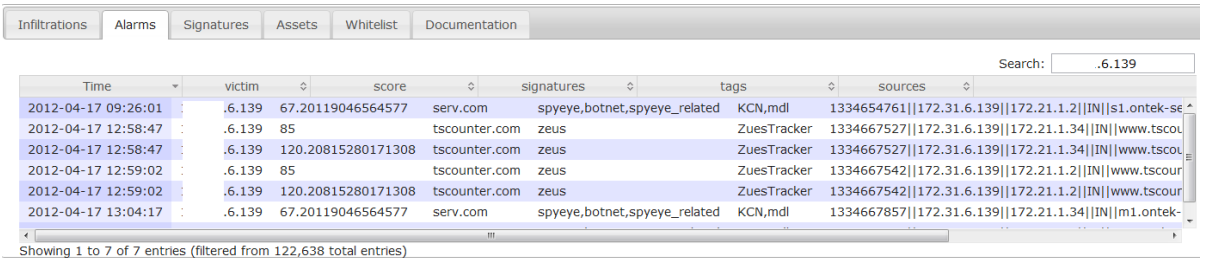
Another possible strategy for finding infected hosts is looking at the alarms table with the highest risk score. Figure 20 below list alarms sorted by risk score.




| Time | victim | score | signatures | tags | sources |
|---|---|---|---|---|---|
| 2012-04-17 12:59:02 | .6.139 | 120.20815280171308 | tscounter.com | zeus | ZuesTracker |
| 2012-04-17 12:58:47 | .6.130 | 120.20815280171308 | tscounter.com | zeus | ZuesTracker |
| 2012-04-17 12:58:47 | .6.139 | 120.20815280171308 | tscounter.com | zeus | ZuesTracker |
| 2012-04-17 12:59:02 | .6.139 | 85 | tscounter.com | zeus | ZuesTracker |
| 2012-04-17 12:58:47 | .6.130 | 85 | tscounter.com | zeus | ZuesTracker |
| 2012-04-17 12:58:47 | .6.139 | 85 | tscounter.com | zeus | ZuesTracker |
| 2012-04-16 05:56:48 | .222.2 | 85 | aqua-a.sub.jp | zeus | ZuesTracker |
| 2012-04-12 13:34:41 | .222.2 | 85 | aqua-a.sub.jp | zeus | ZuesTracker |
| 2012-04-25 07:40:43 | .21.48 | 76.36753236814714 | arendalfotball.no,194.63.248.43 | zeus | mdl |
| 2012-04-24 07:41:17 | .203.227 | 76.36753236814714 | 213.131.252.251,mitglied.lycos.de | malware,zeus,exploit | mdl |
| 2012-04-19 11:31:46 | .140.149 | 76.36753236814714 | arendalfotball.no,194.63.248.43 | zeus | mdl |
| 2012-04-30 21:52:33 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:52:33 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:42:33 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:42:33 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:32:33 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:32:33 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:22:33 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:22:33 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:12:33 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:12:33 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:02:32 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 21:02:32 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 20:52:32 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 20:52:32 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 20:42:32 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |
| 2012-04-30 20:42:32 | .139.8 | 72.24956747275377 | exitguide.ru,109.70.26.36 | exploit,zeus,botnet | mdl |
| 2012-04-30 20:32:32 | .139.8 | 72.24956747275377 | exitguide.ru,194.85.61.78 | exploit,zeus,botnet | mdl |

Figure 20 Alarms sorted by risk score

From looking at some of the top alarms it is clear that "6.139" is triggering many of the alarms with the highest risk score. To get a better overview of "6.139" figure 21 shows the alarm table filtered by "6.139" and sorted by date.




| Time | victim | score | signatures | tags | sources |
|---|---|---|---|---|---|
| 2012-04-17 09:26:01 | .6.139 | 67.20119046564577 | serv.com | spyeye,botnet,spyeye_related | KCN,mdl | 1334654761\|\|172.31.6.139\|\|172.21.1.2\|\|IN\|\|s1.ontek-se |
| 2012-04-17 12:58:47 | .6.139 | 85 | tscounter.com | zeus | ZuesTracker | 1334667527\|\|172.31.6.139\|\|172.21.1.34\|\|IN\|\|www.tscou |
| 2012-04-17 12:58:47 | .6.139 | 120.20815280171308 | tscounter.com | zeus | ZuesTracker | 1334667527\|\|172.31.6.139\|\|172.21.1.34\|\|IN\|\|www.tscou |
| 2012-04-17 12:59:02 | .6.139 | 85 | tscounter.com | zeus | ZuesTracker | 1334667542\|\|172.31.6.139\|\|172.21.1.2\|\|IN\|\|www.tscour |
| 2012-04-17 12:59:02 | .6.139 | 120.20815280171308 | tscounter.com | zeus | ZuesTracker | 1334667542\|\|172.31.6.139\|\|172.21.1.2\|\|IN\|\|www.tscour |
| 2012-04-17 13:04:17 | .6.139 | 67.20119046564577 | serv.com | spyeye,botnet,spyeye_related | KCN,mdl | 1334667857\|\|172.31.6.139\|\|172.21.1.34\|\|IN\|\|m1.ontek- |

Showing 1 to 7 of 7 entries (filtered from 122,638 total entries)

Figure 21: Alarms table filtered by "6.139" and sorted by time

The domains that "6.139" has been contacted has been reported malicious and related to the botnet Zeus by multiple sources. It is therefore very likely that the client is infected. To confirm the infection with 100% accuracy more data is need. Either must the client be brought in for forensics or maybe proxy logs could reveal more about the infection if they had been available.

These simple tests seems to indicated that the system works as intended and at least some of the compromised computers get a high client risk score. Because a ground truth is missing it is difficult to test the systems more accurately.

## 5.5 Performance of the system

Testing the proto type system in a virtual machine on my laptop it managed to process one month of DNS log in 111 minutes. That means that the proto type can search about 6,3 MB of log data per second on a normal laptop. This illustrates that even the current proto type system is able to handle real world data loads.

In the current proto type a small selection of known open blacklist was selected. In an operational setting the system would have to handle signatures from more blacklist, it is therefore important that the performance scales close to linearly with the number of signatures. The chosen algorithm for the matching engine, the aho-corasic string matching algorithm, is supposed to scale linearly with the number of signatures, but it is reasonable to expect the performance to have an over linear tendency because as the signature set grows so does the size of the finite state machine that the aho-corasick algorithm uses. When the size of the finite state machine grows performance can be degraded because of the memory hierarchy in modern computer. When the state machine no longer fits in L1 and some of it is moved to L2 cache performance will be reduced, if it no longer fits in L2 cache some of it would be moved to L3 cache or system memory and performance will be further degraded.

A couple of simple modification could probably increase the performance a lot. The implementation the aho-corasick algorithm that is used is quite slow compared to for example the implementation in GNU Grep. Changing the aho-corasick implementation would therefore probably yield considerable performance gains. Another simple improvement would be to parallelize the search by starting multiple searches on different log files. This would probably scale linear up to the number of available CPU cores and hard drives.

# 6 Conclusion

In this thesis a proto type botnet detector is built and it is demonstrated that it can be used for detecting botnets in real world scenarios.

The detector is based on four main components. 1) Algorithm for quantifying the risk represented by a signature, 2) An algorithm for whitelisting bad signatures that would create false positives, 3) A matching engine for searching log files with a large signature set and 4) An algorithm for identifying compromised computers by aggregating alarm data.

Each of the research questions will now be addressed

*RQ1: Will the system be able to automatically deal with signatures that generate large amount of false positives?*

A test was performed using a signature set consisting of 174 910 signatures. Searching one day of log data with these signatures resulted in total 724 107 alarms. Applying the whitelisting algorithm to the signatures resulted in whitelisting 573 signatures. Rerunning the same search algorithm on the same log data with the whitelisted signatures set resulted in 8393 alarms. That is an alarm reduction of 99%. Manual inspection verified that most of the whitelisted signatures were obviously benign. Exchanging the simple string search algorithm with the proposed matching engine resulted in a further reduction of the alarms to 6955 alarms, about 17% additional reduction. This proves the stated hypothesis, at least for this test scenario, that 1) a few bad signatures that triggers on normal legitimate traffic generates most of the false positives and 2) that the proposed whitelisting algorithm in combination with the proposed matching engine will be able to automatically remove these signatures, there by significantly reducing the number of false positives.

It is not known exactly what whitelisting techniques each blacklists is already using, but the result seems to indicate that the proposed algorithm is an improvement over what is in operational use today.

RQ2: *Will the system simplify the process of detecting compromised host in the network compared to manual inspecting each alarm?*

Searching one month of log data resulted in total 122 638 alarms. Manually checking each alarm would be very labor intensive. To test the proposed system the two hosts with the highest risk score was tested for infection. The test indicated very strongly that they were infected with the Conficker botnet. A check of the host that had triggered the alarms with the highest risk score seem to indicate strongly that it had been infected by the Zeus Crimeware Kit. This seems to confirm the hypothesis that by using the proposed system it should be possible to identified compromised hosts only be looking at the hosts with the highest risk score and the alarms that represents the highest risk. Because no ground through was easily available the test was not design to answer how many of the total infected computers on the network the system would be able to detect.

Putting all the components together the systems seems to provide a significant improvement over standard string search for finding compromised computers. It also makes it much easier to deal with bad signatures that create many false positives. The improvements come from a combination of whitelisting many of the bad alarms and moving from working with alarms directly to working with aggregated client risk scores. The system is complementary and synergistic to some of the recently suggested system in the research literature like Exposure(Bilge et al., 2011) and Notos (Antonakakis et al., 2010).

# 7 Further work

The proposed system in this thesis is dependent on estimating a quantified risk represented by signatures obtained from freely available blacklist. The systems performance could be improved by improving these estimates and automatically learning new signatures.

One interesting method for learning new signatures is modeling that the domains and IP-addresses the threat agent uses as a graph and learns new malicious networks resources by exploring the graph.

The graph can be initialized with the network resources found in blacklists and the risk score calculated in chapter 3. The graph is then extended using a combination of a historic DNS databases and active DNS queries. The risk score of blacklisted domains is then attributed to neighbor domains and IP-addresses and from this a new and extend black list is created.

For example if *evil.com* and *superevil.com* is blacklisted with a risk score of 10 and 20. A historic DNS database is queried to find all IP-addresses these domains has ever pointed to. The IP-addresses are added as nodes to the graph and an edge is created between the domain and IP-addresses.

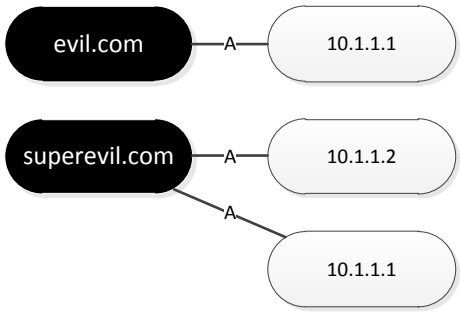



Figure 22 Domain-IP realtions

Further the domain name server of the domain are located using the same history DNS database and is added to the graph.

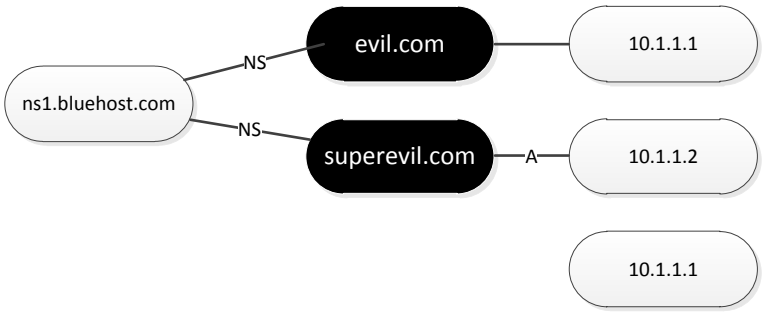



Figure 23 Domain- domains server relations

If the blacklists contain IP-addresses, the DNS database is queried for domains that have ever pointed to them. For example if the IP-address 132.2.1.1 is blacklisted and the domains bad.com and verybad.net has been pointing to it they are added to the graph.

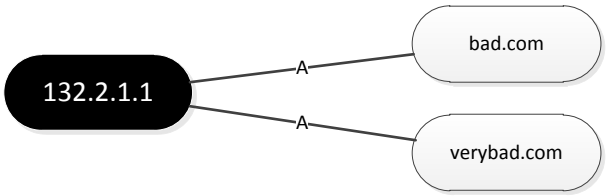



Figure 24 IP-adress-domain relations

When the graph construction is finished, the graph consists of a set of black nodes that are the initial blacklisted domains and a set of white nodes that are in some way associated with the blacklisted domains. The next step is then to run an algorithm on the graph that iterates over the white nodes and calculates a risk score for each of them based on their association with the black nodes. The white nodes that get a high risk score is added to new and extended blacklist.

Google as very successfully used their page rank algorithm to calculate the popularity of web pages using a similar model. More research is needed to investigate this possibility.

# 8 Abbreviations

NIDS – Network intrusion detection system

NAT – Network address translation

FFI - Norwegian Defence Research Establishment

RSS – Root Sum Square

DNS – Domain Name System

HTTP – Hyper Text Transfer Protocol

IRC - Internet Relay Chat

TLD – top-level domain

SLD – second level domain

eTLD - effective top-level domain

# 9 List of Tables

# 10 List of Figures

# 11 Bibliography

Aho, A. V., & Corasick, M. J. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, *18*(6), 333-340. ACM. doi:10.1145/360825.360855

Amit, K. (2011). A Wide Scale Survey on Botnet. *International Journal of Computer Applications*. Retrieved from http://www.ijcaonline.org/archives/volume34/number9/4126-5948

Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., & Feamster, N. (2010). Building a Dynamic Reputation System for DNS. *America* (pp. 1-17). USENIX Association. Retrieved from http://www.usenix.org/events/sec10/tech/full_papers/Antonakakis.pdf

Bilge, L., Kirda, E., Kruegel, C., & Balduzzi, M. (2011). Exposure: Finding malicious domains using passive dns analysis. *Proceedings of NDSS*. Retrieved from http://www.cs.ucsb.edu/~chris/research/doc/ndss11_exposure.pdf

Canavan, J. (2005). The Evolution of Malicious IRC Bots. *Proceedings of Virus Bulletin VB* (pp. 104–114). Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.4045&rep=rep1&type=pdf

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), 1-58. doi:10.1145/1541880.1541882

Denning, D. E. (1987). An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, *SE-13*(2), 222-232. doi:10.1109/TSE.1987.232894

*Globalization of Crime: A Transnational Organized Crime Threat Assessment*. (2010). (p. 310). United Nations. Retrieved from http://www.unodc.org/documents/data-and-analysis/tocta/TOCTA_Report_2010_low_res.pdf

Gu, G., Perdisci, R., Zhang, J., & Lee, W. (2008). BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. *Security*.

Gu, G., Porras, P., Yegneswaran, V., Fong, M., & Lee, W. (2007). BotHunter: detecting malware infection through IDS-driven dialog correlation. In N. Provos (Ed.), *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (pp. 1-16). USENIX Association. Retrieved from http://portal.acm.org/citation.cfm?id=1362903.1362915

Gu, G., Zhang, J., & Lee, W. (2008). BotSniffer : Detecting Botnet Command and Control Channels in Network Traffic. *Technology*, *53*(1), 1-13. Citeseer. doi:10.1.1.110.8092

Halpern, J. Y., & Tuttle, M. R. (1993). Knowledge, probability, and adversaries. *Journal of the ACM*, *40*(4), 917-960. doi:10.1145/153724.153770

Hanseth, O., Monteiro, E., & Hatling, M. (1996). Developing Information Infrastructure: The Tension Between Standardization and Flexibility. *Science, Technology & Human Values*, *21*(4), 407-426. doi:10.1177/016224399602100402

Hawkins, D. (1980). *Identification of Outliers (Monographs on Statistics & Applied Probability)* (p. 188). Springer. Retrieved from http://www.amazon.com/Identification-Outliers-Monographs-Statistics-Probability/dp/041221900X

*Introduction to Information Retrieval [Hardcover]*. (2008). (p. 496). Cambridge University Press; 1 edition. Retrieved from http://www.amazon.com/Introduction-Information-Retrieval-Christopher-Manning/dp/0521865719

Karresand, M. (2002). *A proposed taxonomy of software weapons*. *No. FOI*. Linköping University. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-1512

Khan, S., Madden, M., Coyle, L., & Freyne, J. (2010). *A Survey of Recent Trends in One Class Classification*. (L. Coyle & J. Freyne, Eds.) (Vol. 6206, pp. 188-197). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-17080-5

Li, C. (2009). Botnet: Survey and case study. *Innovative Computing, Information and …*. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5412718

Lin, J.-ling, Wang, X. S., & Jajodia, S. (1998). Abstraction-Based Misuse Detection: High-Level Specifications and Adaptable Strategies. In Proceedings of the 11th Computer Security Foundations Workshop. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.8529

Moore, R. a., Kewley, D. L., Parks, R. C., & Tinnel, L. S. (2001). The Information Battlespace preparation experiment. *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, *1*, 352-366. IEEE Comput. Soc. doi:10.1109/DISCEX.2001.932230

Ollmann, G. (2009). Botnet communication topologies. *Retrieved September*, *30*, 2009. Retrieved from http://www.damballa.com/downloads/r_pubs/WP Botnet Communications Primer (2009-06-04).pdf

Robot. (2012).*Encyclopedia Britannica Online*. Retrieved June 2, 2012, from http://www.britannica.com/EBchecked/topic/505818/robot

Sommer, R., & Paxson, V. (2010a). *Outside the Closed World: On Using Machine Learning for Network Intrusion Detection*. *2010 IEEE Symposium on Security and Privacy* (pp. 305-316). IEEE. doi:10.1109/SP.2010.25

Sommer, R., & Paxson, V. (2010b). *Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. 2010 IEEE Symposium on Security and Privacy* (pp. 305-316). IEEE. doi:10.1109/SP.2010.25

Tygar, J. (2011). Adversarial Machine Learning. *IEEE Internet Computing, 15*(5), 4-6. doi:10.1109/MIC.2011.112

# Appendix A: Source code and signatures

The prototype botnet detector was over 2500 lines of python code in addition to about 500 lines of HTML and JavaScript. The signature set including the original and whitelisted signatures amounts to about 700 000 lines of signature data. It is very inconvenient to print and read that much data on paper. The source code and signatures are therefor provided as a separate ZIP file along with the report.

# Appendix B: Log data

The total log dataset is over 100GB of data. It is not provided with the report because of the practical problems of dealing with such large amount of data. There are also security and privacy concerns with regards to making the data publicly available. If anyone would like to continue the research and need access to the log data contact me at grodaas@gmail.com and we can find a way to provide the log data securely.