# DETECTION OF WEB BASED COMMAND & CONTROL CHANNELS

MARTIN WARMER

November 2011

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

UNIVERSITY OF TWENTE.

## ABSTRACT

Recent malware allows criminals to remotely control computers using Command & Control (C&C) channels. These channels are used to perform criminal activities using infected computers. These activities pose a threat to both the user of the infected computer and other computer users on the network. This threat can be mitigated by detecting C&C channels on the network. In this thesis we attempt to improve the detection capabilities for web based C&C channels. We provide a survey of current C&C channel detection techniques and study the behaviour of web based C&C channels. Based on these results, we propose three new techniques for detecting HTTP and HTTPS based C&C channels. We evaluate these techniques and provide an overview of their detection capabilities.

# CONTENTS

# 1 INTRODUCTION

## 1.1 INTRODUCTION ON MALWARE

Malicious software, also known as malware, has existed for almost as long as computers are around. A lot of effort has been put into stopping malware over the years but malware still remains a problem. Everyday, a huge amount of malware is released. For example, Symantec encountered more than 268 million malware samples in 2010 [31], which amounts to more than 8 samples per second. Keeping up with this number of samples is a challenge.

According to a report by ENISA [40] the motivation of malware creators has shifted from showing off technical skills or trying to gain fame to financial gain. This change also marks a shift in the functionality and sophistication of malware. Traditionally, when a piece of malware like a virus was released the creator only could wait for a message in the media or an anti-virus update, to see that the virus succeeded in infecting computers. With the widespread adoption of the internet, malware is now able to contact its creator after it infects a computer. The attacker can thus monitor and control the spread of the virus. More importantly, the attacker can also remotely control the infected computers. This allows him to profit from his creation by, for example, stealing data from the infected machines.

### 1.1.1 *Botnets*

Botnets consist of computers infected with malware which are called bots. These bots connect to a C&C infrastructure to form a bot network or botnet. The C&C infrastructure allows the attacker to control the bots connected to it. This gives the attacker the ability to use these bots for his own financial gain.

The ENISA report [40] mentions several methods used by criminals to make money using a botnet. Bots can be instructed to steal user data, (financial) credentials or credit card details from the infected computers. This data may be used to empty the victims bank account or impersonate the victim and, for example, take out a loan in his or her name. Bots can be used to visit websites and automatically click on advertisements; thus generating profit for the website owner which is paid per advertisement click. A large group of bots can be used to perform a Distributed Denial of Service (DDoS) attack and bring down a server. This can be used to extort from website owners, who can be asked to pay "protection money". Criminals also sell bot access to other criminals. Access usually consists of the ability to send spam via bots, perform a DDoS attack or gain control of the C&C infrastructure for other purposes.

### 1.1.2 *Targeted attacks*

In the case of a targeted attack the attacker wants to infect a specific target. This is quite different from the regular botnets we have described above, where the criminal is not interested in which machines she infects. The goal

of a targeted attack can be to steal certain data from the target or sabotage target systems. This is achieved by infecting one or just a few computers with malware which contacts a C&C server. The C&C server allows the attacker to remotely control the infected computers. The control functionality can be used to infect other computers or search for documents the attacker is interested in. After the data of interest has been found the attacker gives instructions to exfiltrate the data. The exfiltration usually happens via a channel separate from the C&C channel.

Detecting targeted attacks is much harder than detecting untargeted attacks. The malware is only sent to a few targets, making anti-virus detection unlikely, as antivirus vendors are unlikely to obtain a sample of the malware. Detecting the C&C traffic also becomes harder as Intrusion Detection System (IDS) signatures for malware are unlikely to be available and the C&C infrastructure is less likely to appear on any blocklists. Thus, detection of targeted attacks relies heavily on heuristics or human inspection.

Recently targeted attacks have also been known under the name Advanced Persistent Threats (APT). This is an organized attack with the goal of stealing information, where the attacker has large resources and tries to maintain a persistent presence on the compromised systems. As the description indicates, an APT is a form of a targeted attack and also involves malware controlled via a C&C server.

Operation Aurora is an example of a targeted attack that was published in January 2010. It involved malware infections in at least 34 companies and human rights groups including large companies like Google, Northrop Grumman and Dow Chemical [23]. The goal of this attack seemed to be stealing intellectual property and politically sensitive information from the infected companies and organisation. Ghostnet [26] is another example where malware was used. In this case the malware was used to spy on the Tibetan Government in Exile.

### 1.1.3 *Scope of the malware problem*

Malware is associated with huge economic losses. According to an ITU study [18], the total economic loss attributed to malware in 2006 is estimated to be US$ 13.2 billion in direct damages, with estimates of up to US$ 67.2 billion for indirect and direct damages for 2005 in the U.S. alone. Even a single piece of malware can cause large damages if the wrong computer gets infected. For example, in February 2010 a small marketing firm lost US$ 164,000 due to one computer infected with the Zeus bot [48].

One of the reasons for the huge losses related to malware is the large number of computers that get infected. Microsoft reported [14] that in the second quarter of 2010, it removed bots from 6.5 million computers around the world. In march 2010 the Spanish police arrested three men suspected of running a 13 million pc botnet [29]. One of the three arrestees was caught in possession of 800,000 personal credentials. Given these numbers it is clear that malware constitutes a widespread problem.

### 1.1.4 *Model of operation for bots*

One way to decompose the operation of bots is to split it into two phases: the *infection phase* and the *working phase*.

The *infection phase* requires malicious code to be executed on at least one target computer. This can happen in many different ways. A user might be

tricked into executing a malicious program. This can, for example, happen under the pretence that a certain program is required to view a video or there is an important update for some piece of software. Malicious code can also be executed automatically, for example when an infected USB drive is inserted into a computer. Whereas executing programs is often considered dangerous, viewing documents is usually considered safe, as documents are not supposed to contain any code. However, programs used to open documents often contain bugs which can result in execution of code inside the document. Thus, when an user opens a specially crafted document, a bug can be triggered, allowing malware to be installed on his computer. This whole process can happen in the background without the user noticing it.

The *working phase* begins after a computer has been infected. The malware will contact the C&C server notifying the server that it has been installed and asking for new instructions. In some cases the bot also sends the results of an initial set of instructions which were packaged with the bot. This can, for example, include stealing login credentials and uploading them when first contacting the C&C server, thus providing valuable information even if the bot is quickly removed. The bot will continue to contact the C&C server regularly, asking for new commands or sending the results of a previous command.

### 1.1.5 *Detection & reaction*

Bots can be detected at two different levels, the *network level* or the *host level*.

At the *host level*, bots can be detected during infection. During an infection the malware installs itself on the computer and makes sure it will start again after reboot. To do so, the malware has to write a copy of itself on a local disk. An on-access virus scanner may detect this. Similarly, adding itself to a list of programs which automatically starts may trigger behavioural based detection mechanisms. However, malware can disable virus scanners or use a rootkit to hide itself from virus scanners. This makes it impossible to guarantee detection after malicious code has started running even if the virus is known by the virus scanner.

At the *network level*, bots are harder to detect during the infection phase. Malicious code can be obfuscated and sent over a legitimately used protocol. For example, a pdf document transferred via IMAP as an email attachment is in most cases a legitimate document. However, emails with the same document format and protocol can also be malicious. Thus, detection at network-level would require a virus scanner which scans all documents sent over the network. Systems for scanning all traffic sent over the network have been proposed [56]. However, they can not be used in the case of encrypted connections. Furthermore, they are a lot less efficient than the alternative of scanning all files at the host given the large bandwidth of current networks and limited processing time. Network level detection has the advantage of being able to manage detection at a central location instead of managing detection software on all networked devices.

During the working phase, bots are best detected at network level. At the host level bots can pack the code differently, use random filenames or use other tricks to make infections look different. However, at the network level all bots have to use the same protocol to communicate with a specific C&C infrastructure. Thus detection at network level may be easier as there are fewer variants of C&C protocols than variants of malicious code.

Once malware is detected it can be removed from the infected computer(s) by using specialized removal tools or by reinstalling the computer. This protects users from having their data or credentials stolen by malware. It also helps network administrators as malware generating malicious traffic might cause their internet connection to be blocked or blacklisted.

### 1.1.6 *Network traffic generated by bots*

The network traffic generated by bots can be separated into several categories.

The first category is the *infection related traffic*. This is the traffic generated during the infection of a host. This can for example include the download of malware via a legitimate protocol. However, the traffic can also be absent in case the malware propagates via a physical device like an USB drive.

The second category is *C&C traffic*. This is the traffic generated by the bot when it tries to obtain new commands from the C&C infrastructure. The commands are usually simple and involve only small data transfers. However, the criminal does not want to wait very long before a bot executes his command, therefore the bot frequently has to check for new commands.

The third category is *malicious traffic* generated as a result of the commands received. The traffic generated depends on the command received but can include sending spam, sending DDoS traffic or exfiltrating data from the infected computer.

### 1.1.7 *Network protocol usage of bots*

Current malware generates traffic for a variety of different protocols. A paper that analyses samples submitted to the Anubis platform [19] reports that the most used protocols by malware are HTTP, IRC and SMTP. SMTP is mostly used to send spam, while HTTP and IRC are mostly used for C&C. Symantec reported [31] that in 2010, of all command & control servers they detected, "10 percent were active on IRC channels and 60 percent on HTTP". Thus, HTTP is currently the most used C&C protocol. HTTP and IRC are plaintext protocols, but malware may encode data before sending it in a protocol compliant message. This makes it harder to analyse what is being sent over the network.

A small group of malware uses TLS to encrypt (some of) their communication. In a paper about malware analysed using Anubis [19], 0.23% (796) of the samples used TLS. Interesting to note is that almost all of the TLS traffic is described as HTTPS traffic. Furthermore, in the paper it is noted that most of the samples fail to complete the TLS handshake. This may indicate that the malware does not actually implement TLS, but merely communicates on a port which is normally used for TLS connections.

Usage of TLS has also been documented in the case of advanced persistent threats. In a presentation from Mandiant [25] it was stated that the two most common methods for data exfiltration are via FTP or via HTTPS. These protocols are often used for legitimate traffic and are therefore often available unfiltered in corporate networks. The command & control channel is often separate from the data exfiltration channel and can also use HTTPS. The HTTPS connection can use anything from self-signed, stolen through legitimate certificates [46]. APTs have also been seen which use legitimate HTTPS services for command & control or exfiltration. Examples of such services are Windows live mail, facebook, google talk and msn messenger [46].

Many computers connect to the Internet via a proxy, router or firewall. This allows these computers to make outgoing connections, but receiving incoming connection is often not possible. Malware authors take this into account by making malware connect to C&C infrastructure they can control, instead of connecting directly to the malware. Some networks only allow outgoing traffic to certain ports. For example, it is common to only allow port 25 connections to a designated mail server in order to (be able to) block outgoing spam. Almost all networks allow access to the web over port 80 (HTTP) and port 443 (HTTPS). Thus, malware often uses these ports to connect to the C&C infrastructure.

Encrypted traffic on port 443 has two main peculiarities. First, port 443 is usually not blocked by corporate border firewalls, to allow users to browse the World Wide Web. Secondly, payload-based Network Intrusion Detection Systems cannot monitor HTTPS traffic, as the contents are encrypted. This makes it an ideal C&C channel.

## 1.2 PROBLEM STATEMENT

This research aims at detecting malware-infected desktop computers by passively observing network traffic generated by these computers to and from the Internet. In other words, we aim to detect command and control channels masquerading as legitimate web traffic.

We focus on detecting malware at the network level because this provides a centralized solution. No software has to be installed on the hosts and detection is automatically enabled for all hosts on the network. This is especially useful when users connect their own devices to the network and therefore security measures cannot be guaranteed to be present or up-to-date on such devices.

Current techniques for detecting C&C channels are designed for detecting known malware or large botnets. They are not designed to detect very small botnets or single pieces of malware used in a targeted attack. To detect these threats a detection technique is needed which can distinguish legitimate traffic from C&C channels. The focus is on detecting C&C channels masquerading as web traffic. Web traffic is allowed almost everywhere, whereas other kinds of traffic are often blocked, for example, in corporate environments. Furthermore, traffic is generated to a variety of destinations when browsing the Internet.

Current detection techniques are based on inspection of the contents of network traffic. As malware authors want to evade detection they are likely to use encrypted C&C traffic more often. An obvious choice for an encrypted protocol is to use TLS on port 443, which is used for encrypted web traffic.

### 1.2.1 *Research questions*

Generally, detecting malware by identifying C&C HTTP and TLS traffic is a challenging task as these protocols are used for many different purposes. Users may browse the web, view videos, run web applications and perform automatic updates of their software using HTTP. Detection is even more difficult in the case of HTTPS traffic. Because the traffic is encrypted, no information is available about the contents of the traffic. Detection methods therefore have to rely on indirect information about the content, as the size or timing of packets. This provides much less information thus making de-

tection of C&C traffic more difficult. To the best of our knowledge, there is no detection technique that can detect malware by observing encrypted C&C traffic on port 443 or can generically detect single instances of HTTP C&C traffic.

Therefore, the main research question is:

*How can we distinguish C&C web traffic from legitimate web traffic in both the encrypted and unencrypted case?*

To address the research question, we tackle the encrypted and unencrypted cases separately. We address the first case by selecting and benchmarking different classification and anomaly detection techniques which can distinguish encrypted C&C traffic from legitimate encrypted traffic. We address the second case by designing and benchmarking different anomaly detection techniques which can distinguish C&C web traffic from legitimate web traffic.

By further problem decomposition we therefore extract the following research sub-questions:

1. How can a dataset of C&C web traffic be obtained?
2. How prevalent is the usage of HTTP based C&C channels in malware? What are distinguishing characteristics of HTTP C&C traffic?
3. How prevalent is the usage of C&C channels on port 443 in malware? How prevalent are TLS or SSL C&C channels in malware?
4. Which method works best to distinguish legitimate TLS or SSL traffic from TLS or SSL C&C channels? What detection and false positive rates can be achieved?
5. Which method works best to distinguish legitimate HTTP traffic from C&C HTTP traffic? What detection and false positive rates can be achieved?
6. How do we set-up proper experiments to measure both the "detection rate" and the "false positive rate" of each technique?

### 1.2.2 *Layout of the thesis*

The rest of this thesis focuses on addressing the main research question and sub-questions. In more detail, chapter 2 provides a survey of current C&C channel detection methods. Chapter 3 provides an introduction of the protocols used by web traffic. In chapter 4 we describe how we collect and analyse a dataset of HTTP and TLS based C&C traffic. Based on the analysis, several detection techniques are proposed in chapter 5. An experiment is set-up in chapter 6 to measure both the "detection rate" and "false positive rate" of the proposed techniques. Using these results we evaluate the proposed techniques and answer the research questions in chapter 7.

# 2 CURRENT C&C CHANNEL DETECTION METHODS

Many methods for detecting C&C channels have been proposed. These methods range from signature based detection to automatic correlation of network traffic patterns. This chapter contains a short survey of current C&C channel detection methods.

## 2.1 GENERAL OVERVIEW

Many of the C&C channel detection techniques focus on detecting C&C channels using a specific protocol. Therefore, the detection techniques in this survey are grouped per protocol. In this way we aim at comparing the different approaches taken to detect similar C&C channels. Besides protocol-specific techniques, several protocol-independent techniques are also included. These techniques are grouped by the underlying principle used for detection. In this way we aim at comparing different techniques based on the same principle. For each group we will briefly describe the general working principles, together with the main advantages and disadvantages.

An overview of the detection techniques in this survey can be found in table 2.1. This table also specifies the following attributes for each detection technique.

DESCRIPTION A very short description of what the technique is based on. For more details the reader is referred to the corresponding paragraph or the reference provided.

SETUP DATA The input needed to train or set-up the detection algorithm. Most detection techniques which require training data need legitimate or C&C traffic to built a model of such traffic. Other algorithms require known C&C traffic signatures or blacklists. A large set of techniques require no setup data, relying only on heuristics embedded in the detection algorithm.

GROUP DETECTION Techniques which are group based are designed for detecting groups of hosts with similar C&C traffic. By grouping hosts these methods are able to filter out false positives and thus generate fewer false positives. However, group detection methods are unable to detect single instances of bots, requiring multiple hosts to be infected with the same bot before they can detect it.

GENERIC DETECTION Techniques which are generic are able to detect bots which were unknown during their training or setup phase. Generic detection techniques are useful because new bots are created everyday. Non-generic detection methods require a lot of work to keep up to date for detection of the newest bots and provide a time window during which bots are not detected. However, generic detection methods have the disadvantage that they generate false positives when trying to detect unknown bots.

MALICIOUS TRAFFIC Some techniques rely on detection of malicious traffic for C&C channel detection. Using malicious traffic, these methods re-

| SECTION | REFERENCE | DESCRIPTION | SETUP DATA | GROUP DETECTION | GENERIC DETECTION | MALICIOUS TRAFFIC | PAYLOAD INSPECTION |
|---|---|---|---|---|---|---|---|
| 2.2.1 Blacklisting | [1][6][7][12] | C&C server blacklist | Blacklist | | | | |
| 2.2.2 Signatures | [55] | HTTP signatures generation | C&C traffic | | | | X |
| | [65] | IDS signature generation | C&C traffic | | | | X |
| 2.2.3 DNS | [61] | Non-existent domains | | | X | | |
| | [24] | DNS group activity | | X | X | | |
| | [15] | Reputation score | Blacklists, Domain & IP info | | X | | |
| | [22] | Fast-flux domains | | | X | | |
| | [21] | Encoded DNS replies | Legitimate and C&C traffic | | X | | X |
| 2.2.4 IRC | [35] | Nickname signatures | Signatures | | | | X |
| | [20] | Port scanning per channel | | X | X | X | |
| | [45] | Suspicious host clustering | | X | X | X | |
| | [47][49] | Human/bot distinguisher | Legitimate and C&C traffic | | X | | |
| 2.2.5 HTTP | [66] | Blacklist non-linked URLs | | | X | | X |
| | [42] | Fast-flux websites | | | X | | |
| 2.2.6 P2P | [32] | P2P network detection | | X | X | | |
| | [67] | File-sharing/bot distinguisher | | | X | | |
| | [52] | Model C&C traffic | Legitimate and C&C traffic | | | | |
| 2.2.7 Temporal | [17] | Connection regularity | | | X | | |
| | [34] | Temporal persistence | | | X | | |
| 2.2.8 Anomaly | [63] | 1-gram | Legitimate traffic | | X | | X |
| | [53][16][64] | n-gram | Legitimate traffic | | X | | X |
| | [44] | DFA model | Legitimate traffic | | X | | X |
| | [50] | n-gram clustering | | X | X | | X |
| 2.2.9 Correlation | [36] | IDS event correlation | IDS signatures | | X | X | X |
| | [38] | Group similarity | | X | X | X | X |
| | [37] | Group similarity | | X | X | X | X |

Table 2.1: Overview of C&C channel detection techniques

duce the number of false positives they generate. However, their detection is limited to bots which generate malicious traffic. Thus, bots acting as proxy or stealing data from the local computer are not detected.

PAYLOAD INSPECTION Techniques which rely on payload inspection for C&C channel detection. This has the advantage that more information is available for detecting C&C channels. However, the disadvantage is that payload inspection requires more resources. Furthermore, payload inspection does not help for detection of encrypted C&C channels, as properly encrypted data is indistinguishable from random data.

## 2.2 C&C CHANNEL DETECTION TECHNIQUES

### 2.2.1 *Blacklisting based*

A simple technique to limit access to C&C infrastructure is to block access to IP addresses and domains which are known to be used by C&C servers. There are several blacklists available which contain domain names and IP addresses of C&C servers like the Zeus Tracker [12] or AMaDa [1]. More general blacklists are also available which list sites hosting malware like "malware domain list" [6] and "malware domains" [7].

The advantage of using blacklisting is that it is simple to implement. Furthermore, blacklisting rarely produces any false positives, given that the blacklists are maintained properly. The disadvantage of using blacklisting is that it requires malware researchers to maintain an up to date list of all domains and IP addresses associated with malware. This has two main drawbacks. First, it is expensive to build the blacklist as this requires manual work. Second, the technique creates a "window of opportunity" for attackers during which new C&C servers are not blocked.

### 2.2.2 *Signature based*

A popular technique for detecting unwanted network traffic is to use a signature based Intrusion Detection System (IDS). Such a system tries to match the traffic it observes to descriptions of known unwanted traffic called signatures. This allows the IDS to detect unwanted traffic as long as a signature is available.

Signatures for C&C traffic can be created manually by carefully analysing malware. This is however a very time consuming process, thus techniques for automatically generating signatures have been proposed. Perdisci et al. [55] have proposed a technique to automatically generate signatures for HTTP C&C traffic. The technique clusters similar bot traffic, tokenizes HTTP requests and uses the Token-Subsequences algorithm to generate signatures for each cluster. Wurzinger et al. [65] have proposed another technique to generate signatures by first splitting traffic into snippets likely to contain commands and generating token sequences of these snippets as signatures.

The advantage of signature based detection is that known bot traffic can be easily detected if malware researchers have created a signature. The disadvantage is that bots are often obfuscating or encrypting their traffic which makes it much harder or even impossible to write a signature. Furthermore,

there is always a time window before the signature is released, in which malware can operate without detection.

### 2.2.3 *DNS protocol based*

A bot needs to know the IP address of the C&C infrastructure to communicate. This address can be hard-coded in the bot or it can be retrieved from a domain name. Using a domain name provides more flexibility as it allows the attacker to change the IP address easily.

The domain names requested by a host can be monitored for C&C traffic detection. Villamarín-Salomón and Brustoloni [61] have shown that a host which is repeatedly requesting a domain name which doesn't exist is more likely to contain malware. These requests may indicate that the malware is trying to reach a C&C server which has been taken down. Another indication of C&C traffic proposed by Choi et al. [24] is a group of hosts requesting a new domain name at the same time. This may happen when several hosts are part of a botnet and a new C&C domain is setup, for example, because a previous C&C domain gets taken down. A different approach is taken by NOTOS [15], a system which tries to estimate the likelihood that a domain is being used for malicious purposes. To do this, many features like network information, DNS zones, black- and whitelists are combined to calculate a reputation score.

Maintaining a DNS server and C&C server at a fixed address increases the chance that it will be taken down. Therefore, bot creators have started using fast-flux domains [41]. These are domains for which the owner rapidly changes the IP address to which a domain points and, optionally, the IP address of the DNS server as well. Caglayan et al. [22] have proposed a method for detecting fast-flux domains using multiple DNS responses. They take into account the Time To Live (TTL) of the domain, the number of unique responses seen, and the geographic dispersion of the IP addresses in the responses. By combining this data they are able to detect fast-flux domains in real-time.

Instead of just using DNS to find the C&C server, bots can also use DNS as C&C protocol. Bos et al. [21] have proposed a system to detect DNS based C&C channels based on the entropy of DNS responses. Replies from C&C servers often contain encrypted and encoded data, which has a higher entropy than regular DNS responses.

The advantage of using DNS based systems is that DNS traffic is low bandwidth and low volume, thus only a tiny amount of the total network traffic needs to be analysed. Except for the technique of Bos et al. [21], these techniques don't detect the actual C&C traffic, but the DNS request(s) used to lookup the C&C server. As a consequence, these techniques can not detect C&C channels if the domain looks normal. Thus, bot creators can avoid detection by using a DNS infrastructures similar to a regular webhoster in combination with a legitimate sounding name. Another way they may avoid detection is to not use DNS at all, but use IP addresses to connect to C&C servers.

The detection technique of Bos et al. [21] provides a good method for detecting DNS based C&C channels. However, if only a small amount of data needs to be transferred, detection can be avoided by encoding it as low entropy data.

### 2.2.4 *IRC protocol based*

Traditionally, Internet Relay Chat (IRC) has been used for C&C of botnets. Therefore, several bot detection techniques have been proposed which look for specific features in IRC traffic or try to distinguish human from bot-generated IRC traffic. One of the early examples of such a system is Rishi [35] which scores IRC nicknames using a set of rules to detect C&C traffic.

A botnet detection system has been proposed by Binkley and Singh [20] which clusters computers by IRC channel. This automatically groups bots using the same IRC channel for C&C. For each cluster, the network is monitored to detect TCP SYN scans and a cluster is marked as infected if scanning activity is detected. Thus, the system is able to detect IRC based bots performing TCP SYN scans.

A system by Karasaridis et al. [45] collects flow level data for all hosts which have triggered a suspicious behaviour detection system. By clustering this data, C&C infrastructures can be detected under the assumption that constant IP addresses are used. This allows for detection of bots and the associated C&C server given that the bots are performing suspicious activities which can be detected.

Other approaches have focused on separating IRC traffic into traffic generated by humans and bots. SVM classification on non payload data like packet histogram size and direction has been reported to provide 95% accuracy [47]. Others have used flow data to perform similar classifications achieving high (30-40%) false positive and (10-20%) false negative rates [49].

Focussing on a specific protocol, IRC based techniques can take advantage of specific properties of IRC traffic. This allows them to, for example, group traffic per IRC channel even if the IRC channel is on a server which is also used for legitimate IRC traffic. IRC based detection techniques have been around longer than any other detection technique. This gives them the overall advantage that these techniques have had more time to be improved. The disadvantage is that only IRC based bots can be detected. There are currently many more HTTP based botnets than IRC based botnets, which limits the set of bots that can be detected.

### 2.2.5 *HTTP protocol based*

Xiong et al. [66] have proposed a technique to detect HTTP C&C traffic using user interaction. The technique analyses requested websites statically, to obtain all domains linked to in the page. These domains are added to the whitelist and any requests to a domain not on the whitelist requires user confirmation. It is claimed that after training, this system will only occasionally require users to confirm new domains. The disadvantage of this system is that it requires users to make informed decisions. For example, most users will find it difficult to determine whether a request made by an automatic updater is legitimate. These requests are often generated in the background and may not contain the program or vendor name in the domain. Furthermore, if whitelists are shared between users, the mistake of a single user can allow malware on all computers to access a C&C server.

Hsu et al. [42] have proposed a technique to detect fast-flux hosted webservers. Criminals may use such webservers as C&C server or use them to host phishing sites. Fast-flux servers often use a large collection of bots as proxies which redirect all requests transparently to the actual webserver. The DNS records of these sites are frequently updated to point to a different

bot. Such sites can be detected by measuring the timing of the requests. Bots usually run on slower desktop computers with slower Internet connections. Thus, they will respond slower than an average server. As the requests are redirected, the difference in response time is even larger. A disadvantage of this technique is that it will detect all websites hosted on slower servers. Thus, it may block legitimate websites hosted on a broadband Internet connection.

### 2.2.6 *Peer to peer protocol based*

A small selection of bots uses a peer to peer (P2P) protocol [60] for its C&C channels, instead of using, for example, a HTTP or IRC server-based C&C channel.

François et al. [32] have shown that P2P networks can be detected and distinguished from each other. To detect these networks they generate a graph of hosts talking to each other and calculate the "pagerank" of each node and cluster the nodes. This provides detection of separate P2P networks, but although their work focuses on detecting P2P botnets, they do not provide any method to distinguish legitimate P2P networks from P2P botnets.

Yen and Reiter [67] provide a method to distinguish P2P C&C traffic from file-sharing P2P traffic. The detection is based on the combination of several behaviours observed for C&C P2P channels. They use the volume of data transferred, the peer churn in the P2P network and the timing between requests as distinguishing behaviours.

Noh et al. [52] propose another method to distinguish legitimate and C&C P2P traffic. They collect and cluster flows of both legitimate and C&C traffic. The attributes of these clustered flows are then compressed into a 7-bit state. Using these compressed states, a Markov model is built for each cluster. Peer to peer C&C traffic is detected by comparing the observed traffic with the Markov models of both legitimate and C&C traffic. If the traffic is similar enough to previously seen C&C traffic it is flagged as C&C traffic.

The first two techniques described above provide generic detection of P2P networks, which is a first step for detecting P2P C&C traffic. While the method of Yen and Reiter [67] can distinguish file-sharing and C&C P2P traffic, it may generate false positives for legitimate non file-sharing P2P networks. Thus, it may, for example, detect P2P VoIP calls on the Skype network or P2P money transfers using Bitcoin. The method of Noh et al. [52] avoids this problem, but can only detect P2P C&C traffic which is similar to the P2P traffic which was used for training. A disadvantage of these techniques is that malware may avoid detection by (ab)using a legitimate P2P network.

### 2.2.7 *Temporal based*

A bot regularly has to send traffic to the C&C server in order to able to receive new commands. Such traffic is sent automatically and is usually sent on a regular schedule. The behaviour of user-generated traffic is much less regular, thus bots may be detected by measuring this regularity. AsSadhan et al. [17] have proposed a system to detect hosts generating regular traffic. The system divides time into fixed size timeslot and records the address and packet count of each timeslot. A periodogram of these counts is calculated which shows regular traffic as a peak. If a significant peak is found, the traffic is flagged as regular.

Giroire et al. [34] have designed a detection system which measures the temporal persistence of traffic. The system attempts to find hosts which keep connecting to the same server. Bots are likely to keep connecting to the same C&C server as long as it is online, thus persistent connections may be used to detect C&C channels. The methods measures persistence at several different time-scales using timeslots. This provides detection for regular connections even if the timings are randomized within an interval.

The advantage of using a timing based approach is that it is protocol independent, requiring only that the bot regularly sends or receives traffic from a C&C server. However, legitimate software may also sent regular requests, leading to false positives. These legitimate regular connections need to be filtered using a whitelist, which may, for example, contain regular checks for new mail or software updates. The disadvantage of time based detection systems is that a significant amount of work is required to maintain a whitelist.

### 2.2.8 *Payload anomaly detection*

Payload Anomaly detection is based on the assumption that it is possible to build a model of legitimate traffic content. Any network traffic not conforming to this model is considered anomalous. Given a perfect model of legitimate traffic such a system should be able to detect all C&C and other non legitimate traffic. However, practical systems are limited by the modelling technique chosen and the availability of representative legitimate data.

One of the early systems proposed is PAYL [63] which builds a model based on the byte frequency distribution of the traffic contents. For every combination of traffic length and port number, a model is generated. This diversifies the model to support multiple protocols and request types. However, mimicry attacks have been published [30] which allow an attacker to build an attack with the correct distribution. More advanced models have been proposed which try to model the distribution of $n$-grams efficiently for $n > 2$. Approximations of $n$-gram probability distributions have been made using Support Vector Machines (SVM) [53], Hidden Markov Models (HMM) [16] and bloom filters [64]. Modelling the data using $n$-grams for $n > 2$ makes mimicry more difficult and allows the model to include keywords used in the protocols monitored. Detection techniques based on tokenization of the data contents have also been proposed using, for example, Discrete Finite Automata (DFA) to build a model of legitimate HTTP traffic [44].

Lu et al. [50] have proposed an anomaly detection method using $n$-grams for C&C channel detection. It is based on the assumption that the content C&C traffic is less diverse than the content of legitimate traffic. The detection methods first classifies all traffic into application groups using signatures or a $n$-gram based decision tree. For each session, the temporal $n$-gram distribution is computed, which is the $n$-gram distribution of all traffic in a fixed time window. The resulting distributions are clustered and the cluster with the smallest standard deviation is considered the botnet cluster. Thus, all sessions in that cluster are marked as C&C traffic. This technique provides a method for detecting groups of hosts using the same C&C protocol. It has been evaluated for IRC based C&C channels and has shown good performance for detecting such channels. Its performance on other protocols is currently unknown, but it may generate false positives for automatically

generated traffic. If this traffic is generated by the same software it would have a very low diversity, thus making it likely that it is detected.

The biggest advantage of using anomaly detection techniques is that they are capable of detecting new C&C channels. However, building a good detection model can be difficult, especially when C&C traffic uses the same protocol and keywords as legitimate traffic. Most of the current anomaly detection system are designed to detect attacks on servers. Thus, their performance for detecting C&C channels remains an open question. The technique proposed by Lu et al. [50] has shown good performance for detecting IRC based C&C channels. However, its performance for other protocols is unknown. Furthermore, it can not detect single bots as it is a based on detecting groups of bots.

Bothunter [36], described in section 2.2.9, uses $n$-gram based anomaly detection as part of its detection system. As the standalone performance of this anomaly detection system is not discussed in the paper about Bothunter, it is not included in this section.

### 2.2.9 *Correlation based*

One method to reduce the number of false positives for bot detection is to require several correlated events before raising an alert. This allows the system to use events which by themselves have a high false positive rate. However, by requiring multiple events the system is able to filter out most false positives. The events may be correlated for a single host or for a group of hosts.

Bothunter [36] is a system which combines multiple events for a single host and raises an alert if the events match its bot behaviour model. The events are generated using IDS signatures (see 2.2.2) as well as payload and scanning anomaly detection systems. The payload anomaly detection system is based on an approximation of the $n$-gram distribution of the traffic. Once a single host has generated a sufficient sequence of events which match the behaviour model, it is flagged as a bot infection by Bothunter.

In a botnet, multiple bots are controlled via the same C&C server. Correlation between hosts can provide a way to detect a group of bots connected to the same C&C server. Botsniffer [38] is an example of such a system which groups hosts logged on to the same IRC channel or contacting the same webserver. If enough of the hosts in such a group perform similar malicious activities within a certain time-frame, the group is marked as being part of a botnet. Thus, bots are detected using spatio-temporal correlations between hosts. A more generic variant of Botsniffer is called Botminer [37] which works similarly but groups by server IP address. This has the advantage of being more efficient, as well as being able to detect centralized C&C servers independent of protocol.

The advantage of using correlations to detects bots is that there are fewer false positives compared to using just the individual events. At the same time, this can be a disadvantage because stealthy bots, which generate just one or two events, may not be detected. Furthermore, correlation for a group of hosts only works well if enough bots are present within the monitored network. Thus, such a method will not work well within a small network or for detecting bots with a very limited distribution. All three systems require detection of malicious traffic to confirm the C&C channel. Thus, these systems are unable to detect bots which do not generate malicious traffic.

| SECTION | PROS | CONS |
|---|---|---|
| 2.2.1 Blacklisting | Low false positive rate<br>Easy to implement | Needs update for every new C&C server |
| 2.2.2 Signatures | Low false positive rate<br>Easy to implement | Needs update for every new C&C protocol or implementation |
| 2.2.3 DNS | Protocol agnostic detection of suspicious domains | Only detects domain based C&C servers |
| 2.2.4 IRC | Good detection of suspicious IRC usage | IRC getting less popular as C&C channel |
| 2.2.5 HTTP | Protocol specific detection for most popular C&C protocol | Methods are obtrusive or only detect fast-flux C&C |
| 2.2.6 P2P | Can distinguish separate P2P networks | No generic way to distinguish legitimate from C&C P2P network |
| 2.2.7 Temporal | Can detect most C&C channels | Many false positives for legitimate periodic traffic |
| 2.2.8 Anomaly | Can in principle detect all unencrypted C&C traffic | Unclear if it works well for C&C channels |
| 2.2.9 Correlation | Few false positives | Requires multiple detection events, may miss malware |

Table 2.2: Overview of C&C channel detection technique groups

However, a bot which does not generate malicious traffic may still have a significant impact by, for example, stealing online banking credentials.

## 2.3 DISCUSSION

After presenting the main approaches proposed in scientific publications to detect C&C traffic, we will discuss the pros and cons of each group of detection techniques in this section. We provide an overview of the pros and cons of each group in table 2.2 and go into detail in the rest of this section.

*Blacklisting and signature based techniques* both focus on detecting known C&C channels. They require a large database containing specific C&C servers and C&C protocol implementations, respectively. These techniques provide high detection rates for the C&C channels present in the database, with few false positives. However, maintaining such a database requires a considerable amount of work, as many new pieces of malware are released everyday. Furthermore, there is always a time window in which new C&C channels are not yet in the database, during which the malware can operate without being detected. Despite the disadvantages, these systems are the

most commonly used in practice, due to their low false positive rates. Few false positives make it possible to manually investigate alerts or automatically block traffic without significantly hindering legitimate traffic.

*DNS and IRC based techniques* are the oldest types of technique in this survey that can detect new C&C servers and implementations without requiring updates. Publications of these techniques date back to 2006. Having been public for a long time, these techniques have been improved over the years. This suggests that these techniques might be ready for production use. In the case of DNS based techniques, for example, a commercial product [4] based on NOTOS [15] is already available. From a practical perspective, the IRC based techniques are not as interesting as DNS based techniques, as their scope of use is limited by the scarceness of IRC based C&C channels [31].

*HTTP based techniques* are still very limited in their detection capabilities. Current techniques either require user interaction or limit detection to fast-flux hosted C&C servers. In the first case, the user is asked to decide whether a request is legitimate or C&C traffic when an alert is generated. Most users will find this difficult, as they have no understanding of the technical details of HTTP. In the second case, a large group of C&C servers is ignored by limiting detection to fast-flux servers. The limitations of current techniques make it unlikely that they are usable as practical C&C channel detection techniques.

*P2P based techniques* show the ability to detect separate P2P networks. However, their ability to distinguish legitimate and C&C P2P networks is still limited. Current techniques can only distinguish file-sharing and C&C networks or detect the P2P behaviour of specific malware. In the first case, false positives for legitimate non file-sharing P2P networks are a problem. In the second case, a database of known C&C P2P behaviour needs to be maintained, leading to similar difficulties as for signature based techniques. Thus, current techniques provide a starting point for C&C channel detection, but still have significant weaknesses that prevent them to be successful in practice.

*Temporal based techniques* provide a good detection rate for C&C channels. However, they generate many false positives for legitimate software with periodic behaviour. A whitelist may be used to filter out these false positives. However, creating such a whitelist takes a significant amount of work, as all periodic traffic has to be checked before it is added to the whitelist. As new software and servers are installed, new legitimate periodic traffic may appear, requiring the whitelist to be updated. Thus, maintaining the whitelist also takes a significant amount of work.

*Anomaly based techniques* might in theory detect all C&C traffic. However, little research has been published focusing on detection of C&C channels. Thus, it remains unknown how well anomaly detection works for C&C channel detection.

*Correlation based techniques* have the big advantage that they generate few false positives. In order to detect malware, several detection techniques need to raise suspicious events, which can then be correlated. The suspicious events for the described techniques are generated using port scanning detection, payload anomaly detection and IDS signatures. These event generation techniques have a focus on detecting malicious traffic, thus, malware which does not generate such traffic is unlikely to be detected.

## 2.4 RESEARCH DIRECTIONS

In this chapter, we have described many different C&C channel detection techniques and discussed their advantages and disadvantages. Based on this discussion, we derived the focus of this thesis.

We focus on detecting HTTP based C&C channels because 60% of C&C channels are HTTP based [31] and current techniques have significant disadvantages. None of the current techniques is able to detect the majority of C&C traffic without active participation of users. Thus, we focus on designing a HTTP based detection technique which can detect the majority of C&C traffic passively.

Closely related to HTTP is HTTPS, which secures HTTP traffic by wrapping it in a TLS session. HTTPS traffic has two main peculiarities. First, it is usually not blocked by (corporate) firewalls as it is needed to browse the world wide web. Second, payload inspection based detection systems cannot monitor HTTPS traffic, as the contents are encrypted. Switching from HTTP to HTTPS requires few changes for malware authors, thus if payload inspection based systems are deployed, HTTPS might become a popular C&C protocol. The temporal detection techniques are the only techniques which may be able to generically detect HTTPS C&C traffic. However, the large number of false positives they generate makes it unlikely that these techniques will be used in practice. Therefore, we focus on designing a detection technique for HTTPS based C&C channels.

Anomaly detection systems might be able to detect all types of C&C traffic. However, little research has been published regarding the C&C channel detection capabilities of current anomaly detection systems. Thus, we decide to evaluate whether anomaly detection techniques can be used to create a good C&C channel detection method.

# 3 PROTOCOL INTRODUCTION

In this chapter, we give a short explanation of how HTTP and TLS work. This explanation is intended to provide some basic insight in how these protocols work. It is, however, not intended to be a complete description of either protocol. The exact specification of both protocols can be found by looking up the respective RFCs. If the reader is familiar with both protocols (s)he can skip to chapter 4.

## 3.1 HTTP

The HyperText Transfer Protocol (HTTP) is a text based network protocol designed for retrieval of webpages. To retrieve a webpage the client has to send a request to the server. The server replies with the (dynamically generated) file specified in the request.

A HTTP requests consists of a request line followed by one or more headers. The request line consists of three parts: the method, the URI and the protocol version. A standard request for the root document looks like `GET / HTTP/1.1`. This request specifies the get method to retrieve the file associated with the URI `/`. The request line is followed by one or more headers which are specified as a line containing `Name:Value`. The headers are interpreted based on their (case-insensitive) name thus their order is not significant. Only the Host header is a required request header in the HTTP 1.1 specification. All other headers are optional, but most HTTP clients include many other headers. A description of a few common headers is given below.

ACCEPT specifies the mime-types the client is able to handle

ACCEPT-ENCODING specifies the encodings the server may use in the reply

CONNECTION specifies whether the server should close the connection after sending the reply

HOST specifies the domain name for the request

USER-AGENT contains a string identifying the client version and name

## 3.2 TLS

Transport Layer Security (TLS) is a protocol that provides a secure communication channel between two parties. It can provide authentication of both parties and ensures confidentiality and integrity of the data transferred. TLS has been designed to provide these features for any protocol which can run over TCP. Thus running HTTP over TLS (a.k.a. HTTPS) provides a secure version of the plaintext HTTP protocol.

Authentication of the parties is possible using (x.509) certificates. Usage of these certificates is optional and in practice the server always provides a certificate, while it is rare for a client to provide a certificate. After the server has been authenticated, the client can communicate securely with the server and can, for example, provide a username and password over the secured connection.

Netscape started development of what would become TLS under the name Secure Sockets Layer (SSL). SSL 2.0 was the first version released in 1994, quickly followed by SSL 3.0 in 1995 because of security problems. After SSL 3.0 was released Netscape handed over development to the Internet Engineering Task Force (IETF) which continued development under the name TLS. TLS 1.0 is the result of the continued development of SSL 3.0. It is backward compatible and internally uses the version number 3.1 as protocol version. The term TLS in this thesis should be interpreted as referring to both SSL (3.0) and TLS.

### 3.2.1 *Handshake*

A TLS sessions starts with a handshake during which authentication takes places and session keys are exchanged. The client begins by sending a *ClientHello* message. This message contains a random number and specifies which TLS version, ciphersuites and compression methods the client supports. A ciphersuite is a combination of a key exchange, an encryption and a MAC algorithm for use in TLS. (TLS 1.2 ciphersuites also include a pseudo random function.)

The server chooses a TLS version, ciphersuite and compression method from the *ClientHello* message to use for the session. A *ServerHello* message is sent as response which contains another random number and the selected TLS version, ciphersuite and compression method. The server may optionally send *Certificate*, *ServerKeyExchange* and *CertificateRequest* messages followed by a *ServerHelloDone* message. The *Certificate* message can be used by the server to authenticate itself using a x.509 certificate. If an ephemeral key exchange algorithm is selected, a *ServerKeyExchange* message is sent containing key material from the server. If the client is required to authenticate itself using a certificate, the server sents a *CertificateRequest* message.

Now the client has all data needed to complete the key exchange, compute the secret keys and sent the *ClientKeyExchange* message. The contents of this message depend on the key exchange algorithm which is designed such that only the server can use the message to determine the secret keys. If the server requested a client certificate the client sends its certificate in a *Certificate* message before the *ClientKeyExchange* message which is followed by a *CertificateVerify* message for signing certificates. After these messages the client sends a *ChangeCipherSpec* message. This indicates that from that point on, all messages from the client will be encrypted and authenticated using the secret keys. The first encrypted message is a *Finished* message containing data the server uses to verify that the key exchange and authentication were successful.

The server uses the *ClientKeyExchange* message to compute the secret keys. It sends a *ChangeCipherSpec* message indicating that from that point on, all its messages will also be encrypted. This message is followed by an encrypted *Finished* message to allow the client to verify that the key exchange and authentication were successful.

Besides the complete handshake described above, a shorter handshake, also known as a resumed handshake, is possible. During the complete handshake the server may send a session identifier in the *ServerHello* message. When the client makes another connection shortly after the first one it may include this session identifier in its *ClientHello* message. If the server still has the keys corresponding to the session identifier it will shorten its handshake. It immediately sends the *ServerHello*, *ChangeCipherSpec* and *Finished*

| CLIENT | | SERVER |
|---|---|---|
| ClientHello | $\longrightarrow$ | |
| | $\longleftarrow$ | ServerHello<br>Certificate*<br>ServerKeyExchange*<br>CertificateRequest*<br>ServerHelloDone |
| Certificate*<br>ClientKeyExchange<br>CertificateVerify*<br>ChangeCipherSpec<br>Finished | $\longrightarrow$ | |
| | $\longleftarrow$ | ChangeCipherSpec<br>Finished |
| Application Data | $\longleftrightarrow$ | Application Data |

* Optional message

Figure 3.1: The full TLS Handshake

messages. The client only has to respond with a *ChangeCipherSpec* and *Finished* message to conclude the handshake. This shortened handshake is more efficient both in computation and traffic generated because no expensive cryptographic operations have to be performed and fewer messages have to be exchanged.

The TLS handshake is transmitted as cleartext with the exception of the *Finished* messages. Thus anyone watching the network can read the messages and for example determine the ciphersuite used. If a TLS session is open for a long time one of the parties may want to exchange new keys and request a renegotiation. This starts the handshake protocol again in the middle of an TLS session. In this case the handshake messages are encrypted using the available keys, thus someone watching the network will not be able to read the messages.

| CLIENT | | SERVER |
|---|---|---|
| ClientHello | $\longrightarrow$ | |
| | $\longleftarrow$ | ServerHello<br>ChangeCipherSpec<br>Finished |
| ChangeCipherSpec<br>Finished | $\longrightarrow$ | |
| Application Data | $\longleftrightarrow$ | Application Data |

Figure 3.2: The resumed TLS Handshake

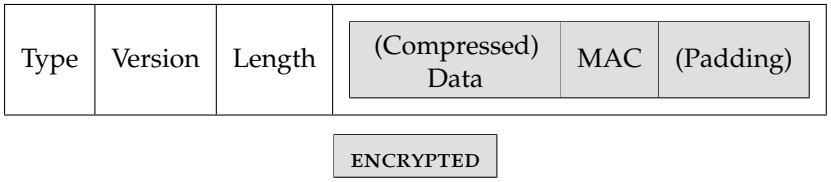



Figure 3.3: TLS Record

### 3.2.2 *Application data transfer*

The record protocol is used to transport all messages in a TLS session. The record protocol provides confidentiality and integrity for all messages sent. Only the initial handshake messages are sent in plaintext as no keys are available before the first *ChangeCipherSpec* message. Application data, for example, HTTP traffic is also protected using the record protocol as this is sent as an *ApplicationData* message.

For every message, the record protocol performs the following operations before transmitting the message as a TLS record. If a compression method has been negotiated during the handshake, the message data is first compressed. A MAC is added to allow the recipient to check the integrity of the data. The data and MAC are padded to a multiple of the blocksize if a blockcipher is used for encryption. The result is encrypted using the negotiated cipher. The encrypted data is put in a TLS record with the type of message, the TLS version and the length of the encrypted data in plaintext. The resulting record is sent over the network.

All data sent using the record protocol is encrypted, authenticated and possibly compressed. The encryption makes sure that no one except the other party is able to read the messages. While the MAC makes sure that the record they receive has been sent by the other party.

### 3.2.3 *Observable features*

Even though all application data is encrypted before transmission some information can still be determined by observing TLS traffic. The messages for the initial handshake are sent in plaintext. Thus any observer can read the *ClientHello*, *ServerHello* and *Certificate* messages. The *ClientHello* message is interesting because it contains the ciphersuites, compression methods and TLS extensions supported by the client. This may be used to (partially) identify the client implementation as different implementations support different ciphersuites and compression methods.

The *Certificate* message(s) contain the certificate(s) used for authentication. In practice the server always authenticates itself, thus its certificate is available. This certificate may be used to determine the host name and organisation of the server. However, as the IP address of the server is already known, this may provide little information. The certificate also contains a signature of the Certificate Authority (CA), which may be used to check the validity of the certificate. Client certificates are rarely used in practice, but if they are used, they may be used to uniquely identify the person or computer on the clientside.

The *ServerHello* message contains the TLS version, ciphersuite and compression method used for the TLS session. This information is useful for correctly interpreting records containing application data. Only the length of each encrypted application data record is available as plaintext. This length

is made up of the length of the (compressed) data, the MAC and any padding. (In TLS 1.1 and later an Initialization Vector may also be part of the length.) The ciphersuite specifies the MAC algorithm used and thus the size of the MAC. In case the ciphersuite specifies a stream cipher no padding is used and the size of the (compressed) data can be obtained by simply subtracting the MAC length from the record length. As compression is implemented in very few clients this will usually give the exact length of the application data. In case a block cipher is used an approximation of the application data length can be made. The padding is allowed to be up to 256 bytes long, but for efficiency reasons the padding is usually between 1 byte and the blocksize long (8 or 16 bytes). Thus if a block cipher is used the length of the application data rounded up to the nearest blocksize can usually be determined.

Besides observing the length of application data, the timing and direction of *ApplicationData* records can be observed as well. This may allow for general identification of the type of application data transferred. An interactive shell may, for example, sent small records for every key press and maintain a long-lived connection which often changes direction as typed characters are echoed back. Whereas web traffic (HTTPS) sessions are usually short-lived with larger records.

# 4 COLLECTING AND ANALYSING C&C TRAFFIC

## 4.1 COLLECTING MALWARE TRAFFIC

No dataset containing C&C traffic is available to study C&C traffic or evaluate detection techniques. Therefore, in this chapter we focus on creating and analysing a dataset of C&C traffic. This dataset is created by collecting malware and running it in a controlled environment where the network traffic can be captured.

### 4.1.1 *Collecting malware*

The first step to creating a dataset of C&C traffic is to collect malware which generates such traffic. This was done by collecting samples for malware families which have been reported to use TLS and by collecting a large set of malicious documents.

Only a small subset of malware uses TLS for its C&C traffic, as was described in section 1.1.7. Therefore, we have decided to focus on collecting malware families for which TLS usage or C&C traffic on port 443 was documented. To this end, we have performed a search on the websites of a variety of anti-virus and security companies to find all malware families for which they have documented usage of TLS or port 443. The resulting list of malware families and usage references is included in appendix A. For each of these families, samples were downloaded from Offensive Computing [9] or from the site documenting TLS usage. Where necessary, these samples were unpacked or decrypted to obtain an executable.

According to Annual Global Threat Report 2009 [13], "the vast majority of modern malware encounters occur with exposure to compromised websites". While the report also notes that PDF files compromised 80% of the web-encountered exploits in $4^{th}$ quarter of 2009. According to Symantec [59], malicious PDF files are not only distributed via websites but are also distributed via mass-mailing or targeted attacks. Thus, malicious PDF documents are an important source of malware infections.

Because malicious PDF documents are a significant infection vector for current malware, we have focused on collecting a large set of malicious documents. The malware dropped by these documents is relatively recent and should provide a selection of malware an average user is likely to be infected by. Furthermore, as the documents are relatively recent, they are likely to use HTTP for their C&C channel, as HTTP is currently the most used C&C protocol [31].

The collected set of malicious documents is a combination of malicious PDF and office documents which have been sent via email or offered for download on a website. The set of documents consists mostly of documents obtained from Contagio [3] combined with some documents obtained from other websites. These documents are known to be malicious and thus likely to drop malware when opened. According to the descriptions on Contagio, some of these documents have been used in targeted attacks. As data exfiltration and C&C channels have been documented to use HTTPS in the case of targeted attacks [46], a few of these documents might also use TLS.

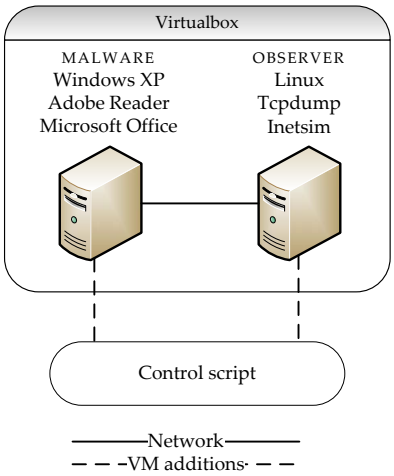



Figure 4.1: Lab setup

While searching malware and malicious documents several forums were encountered were source code of bots was offered. The source code of these bots was downloaded and is analysed in section 4.4.1.

### 4.1.2 *Setting up the lab*

A controlled environment has been setup to run the collected malware samples and capture the network traffic they generate. This environment consists of two virtual machines connected via a virtual network. One machine is used to run the malware, the other is used to observe all network traffic. We refer to the first machine as the *malware machine* and to the second machine as the *observer machine*. The *malware machine* runs Windows XP and is used to run malware executable programs and view malicious documents. To view the malicious documents several major versions of Microsoft Office and Adobe Reader were installed. The use of several major versions increases the possibility of observing malicious behaviour. The malicious code in some documents might depend on a bug in an older version, or simply crash the viewer if not opened with the expected version of the application. All PDF documents were opened using adobe Adobe Reader 6, 7, 8 and 9. All Microsoft Office documents were opened using Microsoft Office 2000, XP, 2003 and 2007.

The *observer machine* runs Linux and provides all network simulation and capturing abilities. An Internet connection is simulated using inetsim [5]. Inetsim provides a range of (fake) servers including DNS, HTTP, HTTPS and SMTP servers. These servers will for example respond to all HTTP and DNS requests generated by samples. All network traffic to and from the *malware machine* is captured using tcpdump [11].

Both virtual machines are controlled via a python script which automatically executes each sample and retrieves the packet captures and inetsim logs afterwards. The virtual machines are isolated and only use virtual machine additions to communicate with the host machine.

### 4.1.3 *Basic analysis of observed network traffic*

Using the lab setup described in the previous section, all collected samples were run and their network traffic was captured. The captured network traffic is analysed in this section.

Out of the sixteen malware families we analysed, only eight produced traffic on port 443 and no traffic was observed on any other port assigned to carry TLS traffic. Only four of these eight families actually use TLS. The other four families use a custom protocol on port 443 and can be easily detected by an IDS which checks for valid TLS records.

Interesting to note is that no TLS traffic was found for most of the obtained malware families, even though these families have been described to use TLS by malware researchers. The most likely explanation for this difference is that the researchers only classified the traffic using port number instead of the actual contents. Another explanation would be that the researchers studied a different variant of the malware than the obtained variants. Most samples were obtained by searching for the malware name thus it is quite likely that different malware samples of the same family were used than the one studied by the researcher mentioning TLS usage.

In total 7842 malicious PDF documents were opened and 47 (0.6%) documents dropped malware which generated traffic on port 443. None of these samples used the TLS protocol, but instead they used HTTP (40% 19 samples) or a custom protocol (60% 28 samples). However, one sample was capable of generating TLS traffic if triggered by opening a web browser (see section 4.4.2).

Out of 301 malicious office documents, 18 (6%) dropped malware which generated traffic on port 443. However, the TLS protocol was not used. Instead, 40% (7) used HTTP and the rest used a custom protocol.

The HTTP traffic generated by the malicious documents was almost all over port 80. Only a few samples used HTTP over port 443 or port 8080 (the alternative HTTP port). Thus, in general the malware in the malicious documents generated protocol compliant requests over the correct port.

The protocols used by the malware were determined using OpenDPI [10]. Which is a tool designed to detect the protocols used on a network using deep packet inspection. After removing the traffic generated by Windows, only HTTP remained as a protocol with significant usage, with 10 out of 16 malware families, 28.2% of office documents and 86.7% of PDF documents generating HTTP traffic. TLS traffic was only observed for the four previously mentioned malware families. Besides HTTP, a further 3 malware families generated SMTP traffic and 2 families generated FTP and IRC traffic. The remaining malware traffic is all classified as unknown and most likely uses a custom protocol for C&C traffic.

## 4.2 ENCRYPTING C&C TRAFFIC USING TLS

As our first observations point out, very few samples currently use TLS for C&C. However, TLS support could be easily added using the standard windows libraries. Malware authors might do this in the near future to evade payload based detection systems. Therefore, HTTP traffic generated by malware samples has been tunnelled over a TLS tunnel to obtain a dataset containing TLS encrypted C&C traffic. This dataset is generated

| FAMILY NAME | NR. OF SAMPLES | PORT 443 USED | TLS TRAFFIC OBSERVED | HTTP TRAFFIC (ANY PORT) |
|---|---|---|---|---|
| Agobot | 49 | No | No | Yes |
| Ghegbot | 19 | Yes | No | Yes |
| Hydraq | 4 | Yes | No | No |
| Mebroot | 56 | Yes | Yes | No |
| Mega-D | 14 | No | No | No |
| NTESSESS | 1 | Yes | Yes | No |
| PingBed | 2 | Yes | Yes | No |
| Pushdo | 50 | No | No | Yes |
| Ramnit.C | 25 | Yes | No | Yes |
| Routrobot | 24 | No | No | Yes |
| Rustock | 28 | No | No | Yes |
| Spybot.worm.gen.p | 50 | No | No | No |
| Spyeye | 26 | Yes | No | Yes |
| Swrort | 26 | Yes | Yes | Yes |
| Tdss | 60 | No | No | Yes |
| Zeus | 54 | No | No[a] | Yes |
| Total | 488 | 8 Yes<br>8 No | 4 Yes<br>12 No | 10 Yes<br>6 No |

*a* Supports TLS but TLS usage was not observed for in the wild samples (see section 4.4.1)

Table 4.1: Summary of traffic observed per malware family

| DOCUMENT TYPE | PORT 443 TRAFFIC | HTTP ON PORT 443 | HTTP (ANY PORT) | NR. OF SAMPLES |
|---|---|---|---|---|
| Office | 18 (6.0%) | 7 (2.3%) | 85 (28.2%) | 301 |
| PDF | 47 (0.6%) | 19 (0.2%) | 6.802 (86.7%) | 7.842 |

Table 4.2: Summary of observed traffic from malicious documents

| PORT | OFFICE DOCUMENTS | PDF DOCUMENTS |
|---|---|---|
| 80 | 68 | 6751 |
| 443 | 7 | 19 |
| 8080 | 4 | 17 |
| Other | 6 | 15 |

Table 4.3: Port usage for HTTP traffic

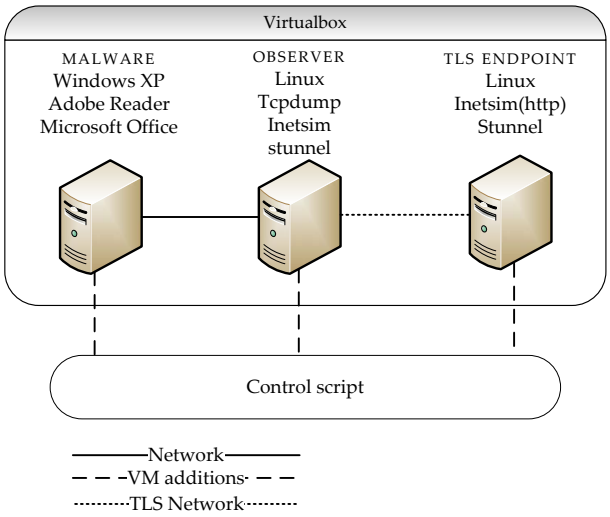



Figure 4.2: Lab setup for TLS tunnelling

using actual malware traffic and provides traffic similar to the traffic if TLS support would be added to the malware.

### 4.2.1 *Lab setup*

The TLS tunnelled C&C traffic is generated using a setup similar to the one described in section 4.1.2. The setup consists of the *malware*, the *observer* and the *TLS endpoint machines* The *malware machine* runs Windows XP and is used to execute the malware. The *observer machine* runs linux and provides network simulation, capturing and TLS tunnelling capabilities. Network simulation and capturing are done using inetsim and tcpdump. The TLS tunnel receives all HTTP traffic and sends it over a TLS connection to the TLS endpoint machine. The *TLS endpoint machine* decrypts the TLS traffic and sends it to its local inetsim webserver. The observer machine captures both the unencrypted HTTP traffic and the TLS tunnelled HTTP traffic for later analysis.

All malware families and documents which generated HTTP traffic during analysis were selected. These samples were opened on the malware machine and the TLS tunnelled traffic was captured. The TLS traffic used the TLS_RSA_RC4_WITH_MD5 ciphersuite which is the ciphersuite preferred by most servers[57]. Compression is explicitly disabled as the only major browser to support it is Google Chrome[1].

### 4.2.2 *Limitations of tunnelling*

Generating TLS C&C traffic by tunnelling HTTP C&C traffic has the disadvantage that exactly the same TLS implementation and configuration is used for all samples. Therefore, this information can not be used to evaluate a detection method, as the detection method may detect the lab setup instead of the C&C traffic itself. The TLS handshake is ignored as its observable attributes are fixed by the lab setup.

The timing of different records in the TLS traffic depends on the virtual network of the lab setup. The timing on the virtual network is likely to be dif-

1 Google Chrome added TLS compression support in September 2010 with Chrome 6

ferent from the timing on a real network. Therefore, timing measurements can not be used for evaluation of detection techniques.

Ignoring the handshake and timing means that only the size application data records can be used for detection. The application data records sent to the server contain the HTTP C&C requests sent by the malware. The content of the replies depend on the configuration of inetsim. Therefore, the replies from the server are ignored. Any detection method evaluated using this data set can thus only use the application data records sent to the server.

### 4.2.3 *Data normalization*

As described in section 3.2.3 the size of application data transferred using TLS can be observed by anyone on the network. In the lab setup no compression and a stream cipher are used, thus the exact size of the application data can be obtained. The application data sent to the server are HTTP requests, thus the size of the HTTP requests generated by the malware can be measured. If the C&C server would use a block cipher the measured size might be up to 16 bytes larger.

Some malware keeps reconnecting to the C&C infrastructure if it doesn't understand the response generated. In the lab setup the responses are generated using inetsim by providing a file of the requested type. Thus, the responses do not contain any recognizable instruction which makes certain malware reconnect many times per second. The large number of TLS sessions generated by this process can significantly change the statistical properties of all sessions. Therefore the sessions are filtered per sample and only unique sessions in terms of initial request size are kept. The result of this process is a list of request sizes for each sample.

## 4.3 ANALYSIS OF TRAFFIC DATASETS

In this section we analyse the collected C&C traffic for both HTTP and TLS (tunnelled) traffic. A network dataset containing legitimate traffic is also introduced. This dataset is used to compare legitimate and C&C traffic attributes and behaviour.

### 4.3.1 *Legitimate traffic dataset*

Network traffic captured at a university network is used as a source of legitimate traffic for evaluation and training purposes. The network capture consists of data generated during a week in a university network. Students and employees are allowed to connect their own devices to this network. The devices connected to the network range from cellphones to computers running a variety of operating systems. These devices run a large range of applications which have generated HTTP and TLS traffic.

It is assumed that the captured traffic contains almost no C&C traffic. Thus, the traffic can be labelled as legitimate when training or evaluating C&C detection methods. While many machines are infected with malware, the percentage of machines infected on any particular network is usually small. Even infected machines usually generate more legitimate traffic than C&C traffic. Thus, if any C&C traffic is present it will be a negligible portion of the total traffic and have very little influence on training or evaluation of detection methods.

### 4.3.2 *Analysis of TLS malware traffic*

The tunnelled malware traffic consists of 16694 TLS sessions generated by 7028 malware samples. These sessions are all very short-lived and consist almost all of a single request, sent by the malware, followed by a reply from the server. There are 34 samples which did not send a request but wait until the server starts the conversation. Another 4 samples tried to send a second request over the tunnel after a reply for the first has been received. However, the webserver we have used does not support multiple requests in a single connection, thus the connection is simply closed after the first reply has been sent.

Most requests fit inside a single TLS record, only 79 samples used more than one record to send their request. Most of these large requests are caused by an upload speed test performed by the agobot malware family. The bot tries to upload around 256 KiB of null bytes to the webserver of an ISP.

As can be seen in figure 4.3a, the HTTP requests sent by malware are small. Half of the requests are smaller than 250 bytes. Although the mean request size is 1140 bytes because of the large speedtest requests sent by the agobot malware family.

The TLS sessions of malware implementing TLS itself have also been captured and normalized. In total 53 TLS sessions generated by 20 different samples have been captured. These sessions are short-lived just like the tunnelled sessions. All initial requests fit inside one TLS record with the exception of the pingbed Trojan which uses 2 TLS records. Compared to tunnelled malware traffic, both the average and median size of the initial request are significantly smaller, with an average of 110 bytes and a median of 193 bytes. One of the reasons for these small sizes is the large number (32%) of sessions with a request size of 0. These are all sessions in which the client does not send a request because the certificate could not be validated. After these sessions the client immediately starts another session in which it ignores the validity of the certificate and sends the request.

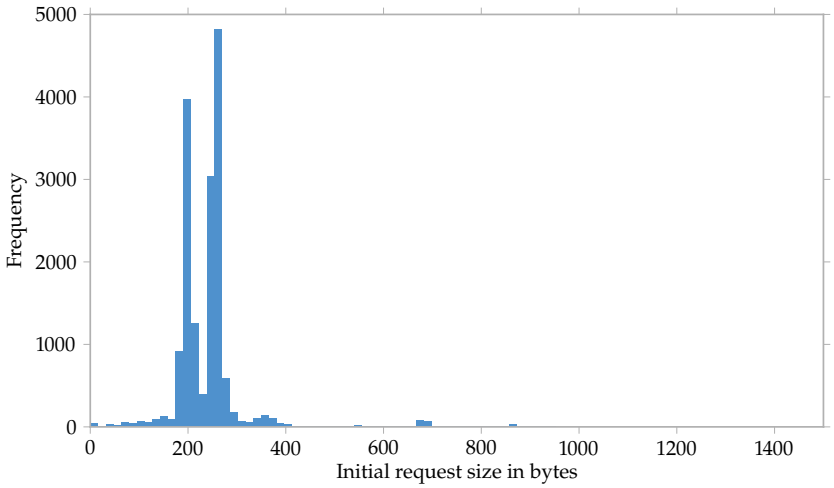
In summary, the dataset has the the following limitations:

- Handshake can not be used
- Only one request per TLS session
- Server reply can not be used
- Request timing can not be used
- Request size can be used

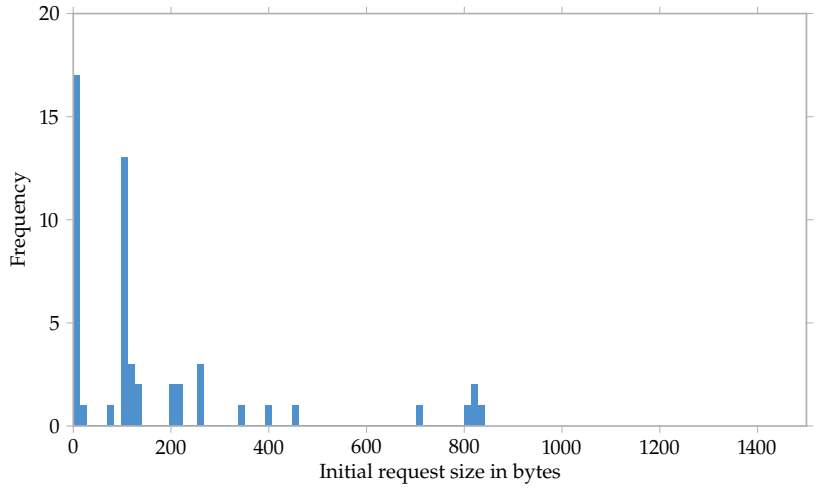
### 4.3.3 *Analysis of legitimate TLS traffic*

Data about all TLS sessions in the legitimate traffic dataset was extracted using the technique described in section 4.2.3 for tunnelled C&C traffic. Due to the limitations of the tunnelled C&C traffic dataset, only the initial request size is extracted, i.e. the size of the application data sent to the server before the first server response is received. These sizes are adjusted for the MAC used and may contain some padding.

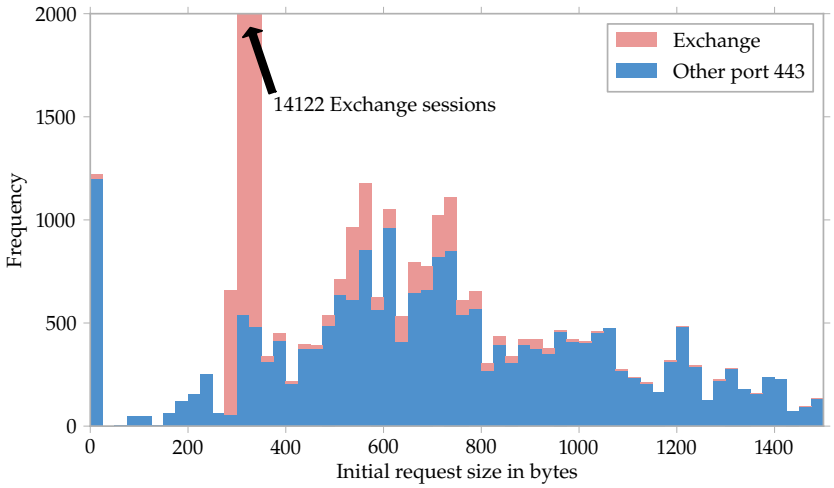
The TLS sessions can be split in different protocols based on port number for HTTPS (port 443), IMAPS (port 993) and POP3S (port 995). Both IMAP and POP3 start with a server welcome message, thus these sessions have




(a) Bytes sent by tunnelled malware in initial request




(b) Bytes sent by TLS supporting malware in initial request




(c) Bytes sent in initial request for legitimate traffic on port 443

Figure 4.3: Initial request size distributions

| STATISTIC (BYTES) | TUNNELLED MALWARE | TLS-AWARE MALWARE |
|---|---|---|
| Median | 250 | 110 |
| Mean | 1.140 | 193 |
| Std. dev. | 14.143 | 275 |
| Min | 0 | 0 |
| Max | 256.772 | 1.267 |
| Nr. of sessions | 16.694 | 53 |

Table 4.4: Distribution of initial request size for TLS C&C traffic

| BLOCK SIZE | CIPHER | USAGE |
|---|---|---|
| Stream | RC4 | 78.0% |
| 8 bytes | 3DES | 1.0% |
| 16 bytes | AES | 20.9% |
| | Camellia | 0.1% |

(a) Cipher usage

| VERSION | USAGE |
|---|---|
| SSL 3.0 | 7.1% |
| TLS 1.0 | 92.9% |

(b) Version usage

| CERTIFICATE | USAGE |
|---|---|
| Valid signature | 97.8% |
| Invalid signature | 2.2% |

(c) Certificate signature validity

| CERTIFICATE DOMAIN | USAGE |
|---|---|
| Matches DNS reply | 99.3% |
| Non-matching DNS reply | 0.3% |
| No DNS reply for IP | 0.4% |

(d) Certificate domain validity

Table 4.5: TLS properties for legitimate traffic

an initial request size of 0 for almost all requests. Upon closer inspection, the HTTPS sessions can be split into two categories. The requests to the university Microsoft Exchange server and the requests to all other servers. The requests to the exchange server make up 31% of all TLS sessions and the other HTTPS traffic makes up 48%. The initial request sizes of both types are significantly different. The Exchange HTTPS requests are small with a median of 338 bytes and a mean of 382 bytes. The other HTTPS requests are much larger with a median of 812 bytes and a mean of 2017 bytes. This difference can be clearly seen in the histogram in figure 4.3c. The number of small Exchange requests is so high that the bar representing them is a factor 7 longer than the largest bar for the other HTTPS traffic. The small requests to the Exchange servers are (most likely) the result of Exchange Activesync clients. These clients run on mobile devices and regularly ask the Exchange server whether new data is available for synchronization. The result of these devices asking for updates every few minutes is that 72% (12327) of all connections to the Exchange server have an initial request size of 300 or 339 bytes.

The captured traffic provides some useful information about TLS usage in practice. The RC4 stream cipher is used to encrypt the data in 78% of the sessions. As this is a stream cipher, the exact size of the application data can be determined for these sessions. AES, triple DES and Camellia block ciphers

| STATISTIC (BYTES) | ALL PORTS | PORT 993 (IMAPS) | PORT 995 (POP3S) | PORT 443 (HTTPS) | PORT 443 WITHOUT EXCHANGE | PORT 443 EXCHANGE |
|---|---|---|---|---|---|---|
| Median | 334 | 0 | 0 | 556 | 812 | 338 |
| Mean | 433 | 0 | 0 | 1.382 | 2.017 | 382 |
| Std. dev. | 342 | 1 | 0 | 65.337 | 83.534 | 1.629 |
| Min | 0 | 0 | 0 | 0 | 0 | 0 |
| Max | 1.499 | 44 | 0 | 12.614.926 | 12.614.926 | 212.095 |
| Nr. of sessions | 55.842 | 4.950 (9%) | 1.727 (3%) | 44.154 (79%) | 27.002 (48%) | 17.152 (31%) |

Table 4.6: Distribution of initial request size for legitimate traffic

are used in 20.9%, 1.0% and 0.1% of the sessions respectively. For these sessions padding is used, thus the measured application data size is slightly larger than the real size. The great majority of the connections (92.9%) uses TLS 1.0 to communicate but a small subset still uses the older SSL 3.0 (7.1%). Compression was only used in one instance, when one client opened several sessions to a specific server. Thus compression does not constitute any problem in practice for measuring application data size.

A total of 544 different certificate chains were used by the servers to identify themselves. Using the root certificates in Firefox and Windows, the signatures of 532 (98%) of these chains could be validated. Leaving 12 (2%) certificates which could not be validated, accounting for 1% of all TLS sessions. These certificates were either self-signed or used a private CA. Thus these servers are mostly likely intended for private usage only. To notable exceptions where a connection to the TOR network which uses randomly generated certificate and a connection to an online backup service which distributes its own CA certificate with its backup software.

Using both TLS and DNS traffic, the domain names on the certificates were also checked. The domain name matches the certificate in 99.3% of the TLS sessions. Incorrect domain names were encountered for 0.3% of the sessions and for 0.4% of the sessions no DNS lookup could be found referring to the server IP address.

### 4.3.4 *Analysis of HTTP malware traffic*

In total, 32197 HTTP requests have been collected, which were generated by 6735 malware samples. The great majority of these requests uses HTTP 1.1 (97.8%). These requests specify to which domain name (or IP address) they are connecting in the Host header. Thus they contain the full URL to which they are connecting. Most requests are GET requests (98.9%), as the lab setup does not contain a C&C server which instructs the malware to upload data.

The size of the HTTP requests has already been analysed in section 4.3.2 as part of the analysis of the tunnelled requests. Thus, the requests generated by malware can be considered small compared to legitimate requests.

The User-Agents of the requests generated by malware provide a clear picture. The User-Agent of Internet Explorer 6 is used by 96.6% of the malware samples. This is the Internet Explorer version installed on the malware machine. Thus, a possible explanation for this User-Agent is that the malware spoofs the User-Agent of the installed version of Internet Explorer. This makes detection of a malicious User-Agent impossible as both Internet Explorer and the malware use the same User-Agent.

We conducted an experiment to see if the malware uses a hard coded User-Agent or spoofs the User-Agent of the installed Internet Explorer version. Internet Explorer 8 was installed on the malware machine of the lab described in section 4.1.2. Because of time constraints only a subset of the samples were executed again on this updated malware machine. The User-Agent of 3658 (96%) out of the 3795 tested samples changed from Internet Explorer 6 to Internet Explorer 8. Given these numbers it is very likely that the majority of the samples spoofs the User-Agent of the installed Internet Explorer version. This User-Agent can be obtained using the `ObtainUserAgentString` function in Windows. Thus, spoofing this User-Agent requires only one or two lines of code.

| USER-AGENT | SAMPLES | |
|---|---|---|
| Internet Explorer 5 | 55 | 0.8% |
| Internet Explorer 6 | 6,521 | 96.6% |
| Internet Explorer 7 | 8 | 0.1% |
| Other | 86 | 1.3% |
| No User-Agent | 83 | 1.2% |

Table 4.7: User-Agents in malware HTTP traffic

| METHOD | MALWARE | LEGITIMATE |
|---|---|---|
| GET | 98.9% | 90.7% |
| POST | 1.1% | 8.5% |
| Other | 0.0% | 0.8% |

(a) Method usage

| VERSION | MALWARE | LEGITIMATE |
|---|---|---|
| HTTP 1.0 | 2.2% | 0.1% |
| HTTP 1.1 | 97.8% | 99.9% |

(b) Version usage

Table 4.8: HTTP properties

### 4.3.5 *Analysis of legitimate HTTP traffic*

The legitimate traffic dataset contains a total of 730,510 HTTP requests generated during a week. The User-Agents of these requests specify a variety of different operating systems and HTTP clients. A quick overview of the operating systems shows that the dataset includes mobile phone traffic and laptops running Windows, Linux and Mac OS X. The majority of the requests are generated by browsers, but for example, SVN also generated requests.

More requests use HTTP 1.1 compared to the malware dataset, but overall the difference is not large enough to provide high detection rates. Similarly the HTTP methods usage in the legitimate dataset is different from the usage by malware, but the differences are not large enough to allow for detection. The User-Agent is specified in 99.4% of all requests, thus a detection method can rely on the User-Agent being specified in the request.

## 4.4 IN-DEPTH MALWARE ANALYSIS

### 4.4.1 *Malware source code analysis*

The source code for several bots was encountered on forums while searching for malware using TLS. Given the forum posts associated with the source code, the intention of sharing it seems to be to allow people to create their own bots. Analysing the source code of bots is much easier than analysing the executables, therefore the source code was downloaded for quick

| NAME | DESCRIPTION | BOT | SERVER |
|---|---|---|---|
| *Blacksun bot* | Only support download and execute via HTTP C&C channel, other commands via custom protocol. | Source | Source |
| *Illusion bot* | C&C channel via either HTTP or IRC supported. Commands include DDoS attacks, proxy servers and e-mailing files. | Builder | Source |
| *PsyProxy* | C&C channel via HTTP allows changing C&C host and updating the bot. Bot provides socks and HTTP proxies. | Builder | Source |
| *Weedbot* | Bot can receive DDoS and update commands via HTTP C&C channel | Source | Source |
| *Zeus* | C&C channel via HTTP or HTTPS. No encapsulation for Socks proxy, RDP and VNC support. | Source | Source |

Table 4.9: Bots with HTTP as C&C protocol

analysis. The collection of bot sources obtained is not representative but is nonetheless interesting as it may provide insight into the C&C implementations used.

The collection of bots contained source code with the oldest samples having timestamps in 2000 and the newest in 2011. This is a long time span but most bots in the collection are at least several years old.

A quick check of the collection indicates that IRC is the most used C&C protocol. This suggests that these are mostly older bots as IRC is becoming less used. Team Cymru reported [58] in January 2010 that "traditional IRC based botnets have remained steady whilst HTTP based botnets continue to steadily climb in number and popularity, doubling in number over 6 months." Symantec reported [31] that in 2010 of all command & control servers they detected "10 percent were active on IRC channels and 60 percent on HTTP".

The set of bot source code contains slightly more than 200 bot(variants) which use IRC. Most of these bots use plaintext IRC as C&C channel. However, a few bots also have the ability to connect to an IRC server using TLS. In all cases they use the openssl implementation instead of the one provided in the windows libraries. Based on the name of zipfiles and directories we derive a rough classification of the TLS-supporting bots in 9 families and 14 variants. However, some families share many similarities, suggesting that code was copied between bots. This suggests that only a small subset of IRC bots employ TLS to encrypt their C&C channel.

Only four bots were encountered using HTTP as C&C protocol. These bots are Blacksun bot, Illusion bot, Weedbot and Psyproxy, which are described in table 4.9.

The only sample supporting HTTP over TLS was the zeus bot. Both an old and recent version of this bot were obtained. The old one has version number 1.1.0.0 and a compilation date in August 2008. This version came complete with source code for the C&C server and a bot builder to configure and generate bots. The recent version of the bot has version number 2.0.8.9

and the last date in the changelog was March 2011. This version came with both the source code of the bot (builder) and the server.

Zeus uses HTTP to communicate with its C&C server. It uses the HTTP implementation provided by the wininet library which is built into Windows. The wininet library also supports HTTPS if the application sets a flag when calling it. The Zeus bot sets this flag if the C&C URL starts with "https". Thus, the bot author has explicitly enabled TLS support in the bot. However, the bot author did not disable the certificate validation which is automatically performed by wininet. This forces the attacker to obtain a valid certificate for the C&C server instead of allowing him to create a self-signed certificate. As a side effect, this also forces the attacker to use a domain name for his server as certificates are not issued for IP addresses.

Besides HTTP, for its C&C channel, the bot also supports other protocols. It can set up outgoing connections to provide direct remote control of the computer via RDP or VNC. It can also set-up a connection to a socks proxy or TCP tunnel to allow access to the (local) networks the computer is connected to. These connections are all made without any obfuscation or encryption. However, the intention to add TLS support in the future can be seen in the source code. The source code contains a complete and working implementation of a TLS socket which is not (yet) used anywhere. This code deliberately supports self-signed certificates making it an obvious method to hide the content of C&C connections from an IDS.

### 4.4.2 *In depth analysis of the samples generating TLS traffic*

The samples generating TLS traffic are analysed in detail in this section. The TLS implementation used by each sample and the protocol used over the TLS data channel are studied.

The samples of the *pingbed* Trojan are hidden inside a PNG image. These images are not malicious in the sense that they make use of any bug in the application that opens them. Instead, these images are simply a way of disguising the executable during download. The images will render fine in a browser and are unlikely to trigger any detection mechanism looking for executables. In order to get infected by this Trojan the computer needs to be running malicious code which extracts the executable from the PNG image and executes it. As the code to do this could not be found, the executables were extracted manually.

The *pingbed* Trojan has to be started with the IP address and port of its C&C server as parameter. When started, it will connect to the C&C server using TLS without verifying the certificate. To do this it uses SChannel, the TLS implementation provided by Windows and disables certificate validation. The Trojan does not send HTTP traffic but uses a custom binary protocol to communicate over the TLS channel. The connection is persistent and allows interactive remote control of the infected computer

The sample of the *NTessess* Trojan is hidden inside a PNG file similar to the one of the Pingbed sample. The executable was extracted manually to perform the analysis. The Trojan uses HTTPS as its C&C protocol. It sends the HTTPS requests using the wininet library in Windows and disables certificate validation. All requests sent by the Trojan are valid HTTP GET and POST requests and the server replies with valid HTTP responses. However, the POST data sent and the replies received are in an obfuscated binary format. The obfuscation seems unnecessary as the requests are sent over an encrypted by TLS connection. However, it may indicate that the Trojan

wants to hide its traffic from a TLS gateway which inspects the traffic or may simply be a feature left over from an earlier implementation using plaintext HTTP.

The sample of *Mebroot* uses the wininet library to setup a HTTPS connection to the C&C server. If the server does not have a valid certificate, the initial connection fails and the malware reconnects, skipping certificate validation the second time. The communication over the TLS channel is valid HTTP traffic and is handled by the wininet library.

The *swrort* sample uses the wininet library to setup a HTTPS connection. However, it tries to connect to an address in the 192.168.0.0/16 range which are private addresses not used on the Internet. Thus, it seems unlikely that the connections are intended to be used for C&C communication. The address is hardcoded and may have been used to test the malware. The TLS connection setup is similar to the mebroot sample. In case the certificate can not be validated a second connection is made. The HTTPS connection is handled by the wininet library.

As described in section 4.1.3 none of the malicious documents we inspected generated TLS traffic when viewed. However, one document did drop malware capable of communicating using TLS, but did not immediately start the malware. Instead, the malware waited for the user to trigger it by opening a browser. This action was performed manually with the result that it tried to connect to mail.google.com using TLS. Using several separate TLS connections the malware programmatically logged on to the webmail interface of Google mail. After logging on, it was only observed to load the inbox and logout. However, another researcher has observed that the malware sent an encrypted email using google mail [62]. The malware uses the wininet library to setup the TLS connection and send HTTP requests. Certificate validation is enabled as mail.google.com should have a valid certificate.

### 4.4.3 *Analysis of metasploit reverse_https traffic*

Metasploit [8] is a popular penetration testing framework which provides tools to attack and remotely control computers. One of the features it provides is meterpreter, a program which allows an attacker to remotely control the attacked computer. Meterpreter can be delivered to the target computer using many methods including server exploits, web browser drive-by attacks or malicious PDF documents. One of the C&C protocols offered by meterpreter is a (reverse) HTTPS connection. As this is an easy to use method for which the source code is available it is worth taking a look at it, as an indicator of how malicious traffic may look.

Metasploit 3.7.2 and 4.0 have different implementations for the C&C HTTPS connection. Therefore, both where separately installed on the observer machine in the lab set-up described in section 4.1.2. An executable containing the meterpreter configured to use HTTPS was generated and executed on the malware machine. The resulting network traffic for both versions is described below.

The meterpreter in Metasploit 3.7.2 uses wininet to send a request to the metasploit server without disabling certificate validation. As the certificate used by metasploit is self-signed this connection fails. After this failed connection, the same HTTPS request is send again with certificate validation disabled. The metasploit server sends the core meterpreter executable as reply and the client executes it. The core meterpreter executable uses openSSL to

setup a new TLS connection over which it communicates using a custom (non HTTP) protocol.

Metasploit 4.0 starts by sending a request to the metasploit server using wininet with certificate validation disabled. The response to this request is the core meterpreter executable which is executed by the client. The core meterpreter regularly sends HTTPS requests to check for new commands and send back the results of the commands.

The meterpreter 3.7.2 HTTPS session stands out in several ways. It uses a self-signed certificate and fails to connect on the first connection. It switches between the wininet and openSSL implementations for connections to the same server. But most importantly, it opens a longterm connection and uses a binary protocol on top of it. Normal HTTPS connections will only be open for a short time to retrieve one or more documents. Thus, a connection in which the data transfer direction switches often or which is open for a long time may be detected as anomalous by an IDS. The meterpreter 4.0 traffic is much harder to detect, it sends valid HTTPS requests using the implementation provided by Windows. The traffic can only be detected by its invalid certificate or the frequent requests required for interactive control.

## 4.5 SUMMARY OF MALWARE OBSERVATIONS

The vast majority of the malware examined uses HTTP as C&C protocol. Whereas only a few samples use TLS to communicate with the C&C server. These samples all use the TLS implementation provided by Windows, with the exception of meterpreter 3.7.2 and the IRC based bots. This implementation is also used by many other programs like Internet Explorer and Windows Update, thus detection of C&C traffic based upon TLS implementation is impossible.

All of the TLS malware allows connections to servers with invalid certificates, with exception of Zeus and the google mail sample. If the servers indeed use invalid certificates this property could be used to detect these samples. Similarly, the double connection attempt in the case of an invalid certificate might trigger detection.

In most cases, the TLS malware uses HTTP to communicate over the TLS channel on port 443. This is the expected protocol for port 443 (the HTTPS port). Therefore, this does not provide extra information to detect TLS malware. However, some samples do not use TLS when communicating on port 443. These can be detected by checking the protocol for traffic on port 443.

The majority of the examined malware uses HTTP based C&C channels. The HTTP requests generated by these malware samples are usually GET requests with a spoofed User-Agent. Where the majority of malware spoofs the User-Agent of the installed Internet Explorer version. Thus, detecting spoofed User-Agents might provide a method for C&C channel detection.

Source code of some bots is freely available online. However, these are mostly older IRC based bots. The source code of more recent bots is rarely available online. Thus, examining bot source code found online does not provide much information about modern bots.

# 5 PROPOSED C&C CHANNEL DETECTION TECHNIQUES

In this chapter, we select three different detection techniques for C&C channels in HTTP or TLS traffic. We leverage existing work in the field of anomaly detection and machine learning to select the detection technique which works best to detect C&C traffic. The selection and evaluation process, which is the main contribution of this thesis, is based on the information and observations in chapter 2 and 4.

## 5.1 MACHINE LEARNING-BASED TLS CLASSIFICATION

As discussed in section 4.3, the distributions of the initial request size of legitimate TLS traffic and C&C TLS traffic are different. These differences should enable us to distinguish legitimate traffic from C&C traffic using machine learning algorithms. Therefore, we propose the following detection technique for C&C channels in TLS traffic based on the size of the initial HTTP request.

### 5.1.1 *Approach*

The general approach of the detection technique is to extract the initial request size from each TLS session, classify it and raise an alert if the session is classified as C&C channel. A graphical representation of this approach can be seen in figure 5.1.

The classifier is generated using a machine learning algorithm trained with the initial request sizes of labelled legitimate and C&C traffic. This should allow the machine learning algorithm to learn the difference between both types of traffic. Thus, the resulting classifier should be able to distinguish legitimate and C&C TLS sessions based on the initial request size. To achieve the best detection we select the initial request size as distinguishing attribute, and we choose the machine learning algorithms.

The size of the initial request sent over a TLS channel can be determined by anyone observing the network traffic. However, as described in section 3.2.3, care has to be taken to take the size of the HMAC, as specified by the ciphersuite, into account.

The proposed algorithm for detecting TLS C&C connections based on initial request size is as follows.

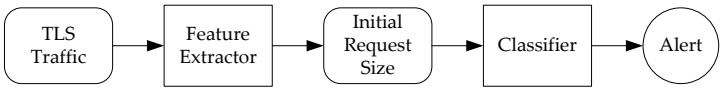



Figure 5.1: TLS based C&C channel detection architecture

*TLS C&C channel training & detection algorithm*

1. For each new TLS session, record
   - the ciphersuite in the ServerHello message;
   - the size of each application data record sent to the server before the server has replied with any application data records.
2. Calculate the initial request size of each TLS session
   a) Use the ciphersuite for the session to look up the size of the HMAC
   b) Subtract the size of the HMAC from the size of each application data record
   c) Sum up the resulting sizes to obtain the initial request size
3. When in *training mode*, build a classifier using initial request sizes labelled as either legitimate or C&C traffic.
4. When in *detection mode*, query the classifier with the initial request size. If the classifier returns the C&C traffic class, then fire an alert.

### 5.1.2 *Details*

The algorithm has to be trained using a labelled set of network traffic. During this training phase, a classifier is build for use in detection mode. This classifier depends on both the legitimate and C&C traffic in the training set. Over time the behaviour of legitimate TLS sessions might change, as new applications and services are introduced. In that case it will be necessary to retrain the algorithm using an up to date set of labelled network traffic.

### 5.1.3 *Selected machine-learning algorithms*

A set of machine learning algorithms has been selected for use with the detection algorithm. Most of these algorithms have been selected because they have been successfully used to detect the protocol used in a TLS session. This indicates that they are able to distinguish between TLS sessions based on the observable attributes of TLS sessions. Thus, while we are not trying to detect different protocols, these algorithms may be able to detect the differences in protocol usage for HTTPS based legitimate and C&C traffic.

A short description and selection motivation of each selected algorithm is included below. For more information about these algorithms we refer the reader to the Weka machine learning toolkit [39], which is the implementation we used for evaluating this detection technique.

ADABOOST is a boosting algorithm which combines multiple classifiers generated by the same machine learning algorithm to obtain a better classifier. In this case we have selected decision stumps to be combined using adaboost to obtain a classifier.
Adaboost has been selected because McCarthy [51] has reported good results using adaboost for detecting the protocol used on top of TLS.

CONJUNCTIVE RULE produces a classifier consisting of a single rule to determine if a TLS session contains C&C traffic.
Conjunctive rule has been selected to test the effectiveness of a simple

classifier. As the distributions of legitimate and C&C traffic are very different a single rule may be enough to achieve good detection.

J48 produces a decision tree as classifier. This is a binary tree containing a comparison at each node and class label at each leaf.
J48 has been chosen because the decision trees it produces can be easily inspected to see how the classifier works. Furthermore, McCarthy [51] has reported good results using J48 for detecting the protocol used on top of TLS.

NAÏVE BAYES uses the Bayesian theorem with an assumed Gaussian distribution of variables for classification. However, in case only one variable is used, the Bayesian theorem can not be used and the classification is based only on the approximated Gaussian distribution of each class.
Naïve Bayes has been selected to test whether a Gaussian distribution can model the initial request size well enough to detect C&C channels.

RIPPER generates a set of rules for classification. These rules are built by combining comparisons using logical AND and OR operators. Classification works by evaluating these rules and returning the class of the first rule which evaluates to true.
RIPPER has been selected because McCarthy [51] has reported good results using it for detecting the protocol used on top of TLS.

SVM generates a classifier based on separation of the classes by a hyperplane. However, as most data can not be separated well using a plane, the input data is mapped into a high dimensional space using a kernel. The hyperplane is constructed and used for classification in this high dimensional space.
SVM has been selected because Dusi et al. [28] have reported good classification results using SVM to detect the protocol used in an SSH-tunnel. While an SSH-tunnel is different from a TLS session, both provide an encrypted and authenticated channel for use by another protocol. Hsu et al. [43] recommend the Gaussian Radial Basis Function as a good kernel function for most data. Therefore a Gaussian kernel has been selected for SVM classification.

## 5.2 SPOOFED USER-AGENT DETECTION

In section 4.3.4 we note that almost all malware spoofs the User-Agent of a legitimate application when sending HTTP requests. We therefore propose a new method for detecting spoofed User-Agents, such a method will be able to detect most malware, but will let legitimate traffic with correct User-Agents pass. We assume that legitimate programs will use their own User-Agent as they have no reason to spoof it.

### 5.2.1 *Approach*

The spoofed User-Agent detection algorithm is based on the header order in each HTTP request. The HTTP protocol allows implementers to send the headers in any order. Therefore, any HTTP client implementation has to (implicitly) fix a header order. As a result, the presence of certain HTTP headers, and their order within an HTTP request are implementation dependent.

We use the implementation dependence of the header order and presence, to build fingerprints of different User-Agents and detect spoofed User-Agents. Our detection method extracts both the User-Agent and the header order from each HTTP request. It looks up the fingerprint for the User-Agent and compares it against the observed header order. If they don't match, an alert is raised. A graphical overview of the detection system can be seen in figure 5.2.

The detection system depends the presence of good fingerprints. A fingerprint consists of a Discrete Finite Automata (DFA) model of the headers, which captures both the presence and order of the headers. Fingerprints are generated by taking a group of requests from the same major version of a HTTP client. The DFA model is built using only the headers which are present in every request. Transitions are added to the model for every pair of consecutive headers.

Fingerprints are only generated for every major version of each HTTP client as the HTTP client implementation is unlikely to change between minor versions. The number of fingerprints is reduced and it is more likely that a fingerprint is available for every request. The fingerprint is generalized by only taking headers present in every request, into account. This makes sure that the fingerprints can be used even if not not every possible set of headers is observed when generating the fingerprint. Furthermore, it helps to avoid false positives if websites add their own headers using javascript and XMLHTTPRequest.

The following algorithm is used to generate fingerprints and takes as input a set of HTTP requests with known correct User-Agents.

*User-Agent fingerprint generation algorithm*

1. Group all requests per major version of each browser (ignoring operating system)
2. For every group:
   a) Extract a list of all observed header orders from the requests
   b) Remove headers which are not present for all requests on the list
   c) Add the states `Start` and `End` to the DFA model
   d) Add every header in the list as a state to the DFA model
   e) For every pair of consecutive headers $m$, $n$ add a transition between $m$ and $n$
   f) For every header $m$ which occurs as first header add a transition between `Start` and $m$
   g) For every header $m$ which occurs as last header add a transition between $m$ and `End`
   h) Save the resulting DFA model as fingerprint

Given a set of fingerprints, the spoofed User-Agent algorithm can attempt to detect requests in which the User-Agent is spoofed. It does this by checking whether the observed header order can be generated by the DFA model for the User-Agent specified in the request. Requests for which no fingerprint is available are ignored, as we have no information about the behaviour of the specified User-Agent. The algorithm for this detection process works as follows.

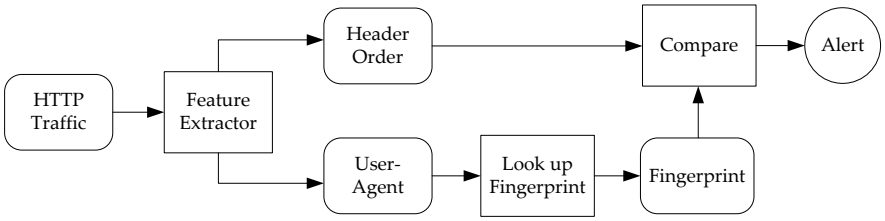



Figure 5.2: Spoofed User-Agent detection architecture

*Spoofed User-Agent detection algorithm*

1. For every HTTP request:
   a) Look up the DFA fingerprint for the User-Agent specified in the request. If no fingerprint is available, ignore the request.
   b) Extract the header order of the request.
   c) Remove headers which are not present in the fingerprint.
   d) Determine whether the header order can be generated using the DFA fingerprint. If this is not the case, fire an alert.

### 5.2.2 *Details*

Identification of web browsers by monitoring network traffic has been attempted before. For example, Yen et al. [68] have demonstrated that browsers can be fingerprinted by classifying flow-data using SVM. The browserrecon project [2] has tried to built a browser fingerprinting database. The database contains a list of known header orders and header values for browsers. Requests are identified by matching the observed header values and order to the values and orders in the database. The implementation requires an exact match of these values, no generalisation is performed. Identification of browsers will thus be very precise but will require a very large fingerprinting database, containing all possible variations of header orders and values. Our approach is different in that it focuses on building a compact fingerprint for identification instead of using exact matches.

The proposed detection technique bears similarities to the DFA based anomaly detection technique described by Ingham et al. [44]. Both methods detect unwanted HTTP traffic and use a DFA model for detection. However, the detection technique we propose is significantly different. We focus on the detection of spoofed User-Agents, instead of attacks against websites. We use headers to build the DFA model, instead of a tokenized input stream. The generalization is performed before building the DFA model instead of on a partially built DFA model. Thus, the contribution of this thesis is in the application of a DFA model for browser fingerprinting and the design of the generalization technique for building the fingerprints.

The obtained fingerprints remain useful for a long period as major releases are rare for most browsers. Thus, new fingerprints have to be generated infrequently. Google Chrome is a big exception as 13 major versions have been released in 3 years. Such frequent updates may be handled by using the fingerprint of a previous version or by automating the process to install a browser and create new fingerprints.

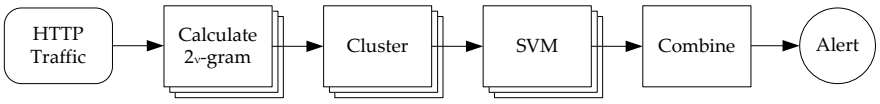



Figure 5.3: $2_\nu$-gram based C&C channel detection architecture

To avoid detection, malware may start mimicking the requests of the spoofed User-Agent. The header order of a legitimate User-Agent can be easily determined by monitoring the requests of the legitimate User-Agent. Once the order is known, the malware can be adapted to generate the same header order for its C&C traffic. Thus, spoofed User-Agent detection only works for malware which does not actively try to evade it. However, spoofed User-Agent detection forces malware authors to make more complex malware capable of mimicry.

## 5.3 $2_\nu$-GRAM BASED ANOMALY DETECTION

In section 2.2.8 we note that payload based anomaly detection systems may provide a method for detecting C&C channels. However, little research has been published about the detection capabilities of payload based anomaly detection systems. Therefore, we propose to re-purpose an existing anomaly detection system for C&C channel detection. Specifically, we propose to use McPaD [54] for C&C channel detection. McPaD is a payload anomaly detection system which approximates $n$-grams using $2_\nu$-grams and which has shown good performance for detecting attacks on HTTP servers.

### 5.3.1 *Approach*

The anomaly detection system attempts to model the $n$-gram distribution of legitimate HTTP requests. However, instead of using the $n$-gram distribution directly, $2_\nu$-gram distributions are used which model the distribution of $(\mathtt{byte}_i, \mathtt{byte}_{i+\nu})$. By using the $2_\nu$-gram distributions for $\nu = 0..n-2$, the $n$-gram distribution is approximated. The advantage of using multiple smaller distributions is that classification can be performed in a space with lower dimensionality, which is more efficient and avoids over-generalization.

In general, the detection system works as depicted in figure 5.3. The $2_\nu$-gram frequency distributions for $\nu = 0..n-2$ are calculated. These distributions are independently clustered and scored using SVM. The resulting scores are combined and if the combined result crosses a threshold, an alert is raised.

Like all anomaly detection systems, this system has to be trained using legitimate traffic. During the training phase, the clustering and SVM classifier are created for every $\nu$ using the following steps. The $2_\nu$-gram frequency distribution of the training traffic is calculated. The $256^2$ different $2_\nu$-grams in this distribution are clustered into $k$ clusters using a feature clustering algorithm proposed by Dhillon et al. [27]. This clustered $2_\nu$-gram distribution is used to train a one-class SVM classifier. The final step is to determine the threshold for raising an alert. The SVM scores of every request are calculated using the trained SVM classifiers. These scores are combined per request using the combination function $f_{\mathtt{combination}}$. The distribution of these combined scores is then used to determine the threshold based on the desired false positive rate $fp_{\mathtt{desired}}$.

The algorithm for training the anomaly detection system using legitimate HTTP requests works as follows.

*$2_\nu$-gram training algorithm*

1. For every $\nu = 0..n-2$
   a) Calculate the $2_\nu$-gram distributions of all HTTP requests
   b) Cluster the $(byte_i, byte_{i+\nu})$ frequencies of the $2_\nu$-gram distributions into $k$ clusters using the feature clustering algorithm proposed by Dhillon et al. [27]
   c) Save the obtained clustering
   d) Build a one-class SVM classifier for the clustered distribution using $\gamma$ as the width of the kernel
   e) Save the obtained classifier
2. Using the obtained clustering and classifiers
   a) For each HTTP request, calculate the one-class SVM scores
   b) Combine the one-class SVM scores using $f_{combination}$
   c) Use the combined scores to compute the best threshold for the desired false positive rate $fp_{desired}$
   d) Save the threshold

The detection algorithm uses the clusterings and SVM classifiers of the training algorithm to generate identically clustered $2_\nu$-gram distributions and calculate the SVM score of each distribution. These values are combined and compared against the threshold obtained during training. If the value is too low, the anomaly detector will raise an alert. This process is performed using the following algorithm.

*$2_\nu$-gram anomaly detection algorithm*

1. For every $\nu = 0..n-2$
   a) Calculate the $2_\nu$-gram distribution of the HTTP request
   b) Cluster the $2_\nu$-gram distribution using the saved clustering
   c) Calculate the score using the saved one-class SVM classifier
2. Combine the scores using $f_{combination}$
3. If the result is below the threshold, raise an alert.

### 5.3.2 *Details*

Ingham et al. [44] have compared different anomaly detection techniques for detecting attacks against HTTP servers. While detecting attacks is different from detecting C&C traffic, this comparison can provide some insight into which anomaly detection methods work best for HTTP traffic. A selection of methods was compared, based on byte distribution, length of request, 6-gram distribution, token-based Discrete Finite Automata (DFA) and Hidden Markov Models (HMM). The 6-gram anomaly detection method achieved the lowest false positive rate at a detection rate of 80%. Given the performance of the 6-gram model for detecting anomalies in HTTP traffic, we de-

| PARAMETER | DESCRIPTION |
|---|---|
| $n$ | $2_\nu$-grams for $\nu = 0..n-2$ |
| $k$ | Nr. of clusters for $2_\nu$-grams |
| $\gamma$ | Width of radial base function kernel |
| $f_{combination}$ | Function to combine SVM scores |
| $fp_{desired}$ | Desired false positive rate |

Table 5.1: Parameters for $2_\nu$-gram anomaly detection

cided to select a method which approximates $n$-grams for C&C channel detection. The McPaD [54] anomaly detection system approximates $n$-grams. It has shown good performance for detecting HTTP server attacks as anomalies, thus it is able to detect anomalies in HTTP traffic. Therefore, we have selected the McPaD anomaly detection system for detection of C&C channels.

There are several variables which can be set when training the anomaly detection system. The $n$-grams approximated can be set by varying $n$ to determine the range of values for $\nu = 0..n-2$. This determines how many $2_\nu$-gram distributions are used for detection. The number of clusters into which $2_\nu$-grams are clustered for a given value of $\nu$ is set by the parameter $k$. Increasing the number of clusters gives the SVM classifier more freedom for classification, but increases computational complexity. The SVM classifier can also be configured by varying $\gamma$, the width of the radial base function used. The function used to combine SVM scores is given as $f_{combination}$. The desired false positive rate $fp_{desired}$ is set to determine the threshold for deciding whether a request is legitimate or C&C traffic.

The result of the training algorithm are the clustering and SVM classifier for each $\nu$ and the threshold for detection. Together, these describe the behaviour of the legitimate HTTP requests in the training dataset. Over time, the behaviour of legitimate HTTP requests may change as new services are introduced. In that case the training algorithm has to be run again using a new set of legitimate HTTP requests.

# 6 EVALUATING DETECTION TECHNIQUES

## 6.1 EVALUATION METHOD

To evaluate the proposed detection techniques, we want to measure both the detection rate and false positive rate of the detection algorithms proposed in chapter 5. The detection rate is the percentage of C&C channels which the technique detects. We want to detect all C&C channels, thus a good detection technique should have a detection rate above 90%. The false positive rate is the percentage of legitimate traffic which is incorrectly detected as C&C channel. This should be as low as possible, as even low detection rates will quickly cause many alerts per minute. Take for example, the HTTP requests collected during a week described in section 4.3.1. With just a 1% false positive rate on this dataset, a network administrator already has to process 3 alerts per minute to process all alerts within a 40 hour working week. Thus, a good detection technique should be able to achieve a false positive rate below 0.1%.

Determining the detection rate using real-life network data can be hard, as this would require us to label all the C&C traffic in advance. Manually labelling all traffic is not possible, as this would take too much time for a reasonably sized dataset. Therefore, the detection rate of the proposed techniques is estimated by measuring how much of the C&C traffic captured in the lab setups of chapter 4 can be detected.

Determining the false positive rate using real-life network data is also not possible because it would require the same kind of labelling. If we simply label all network traffic from a real network as legitimate we are likely to label almost all traffic correctly. The percentage of infected machines on a single network can be expected to be relatively low and even infected machines are likely to generate much more legitimate traffic than C&C traffic. Therefore, the false positive rate is estimated using real-life network traffic, by assuming that it contains no C&C traffic. The dataset described in section 4.3.1 is used for estimating the false positive rate.

The detection and false positive rates are evaluated by classifying the two datasets. However, the same dataset can not be used both for training and classification, as this would bias the results. Therefore, 10-fold cross-validation is used if a dataset is used for training. The dataset is split up into 10 parts. For each part the traffic is labelled as C&C or legitimate using a detector trained on the other 9 parts. This ensures that the detector is tested against the entire dataset while the detector is only used for traffic it has not seen during training

## 6.2 DETECTING TLS C&C TRAFFIC BASED ON INITIAL REQUEST SIZE

### 6.2.1 *Preparing the data for machine learning*

The TLS C&C channel detection technique uses machine learning algorithms to build a model of legitimate and C&C traffic. Using these algorithms requires that the data used for learning exhibits certain properties. The most important property is that the number of malware and legitimate requests

is roughly equal. If the distribution is unequal the model will "optimize" its performance by biasing its output to the majority class. For some machine learning algorithms, outliers can negatively effect the performance of the model built during the training process. To address this problem, all sessions with an initial request size larger than 1500 bytes were removed. This does not remove any C&C channels from the traffic. The only malware that generated requests larger than 1500 bytes is Agobot which generated these requests as part of a speed test and not as part of its C&C traffic.

We want to evaluate whether the detection technique is able to distinguish between legitimate and C&C HTTP requests send over TLS channels. If TLS sessions with other protocols are used, the classifier might distinguish between HTTP and non HTTP traffic instead of between legitimate HTTP and C&C HTTP traffic. Thus, we would like to remove all TLS sessions which do not carry HTTP traffic from the dataset. This is achieved by only keeping TLS sessions on port 443. This port is the designated port for HTTP over TLS traffic, thus the traffic on port 443 is likely to be HTTP over TLS traffic. If malware uses a different port for its HTTPS C&C traffic, techniques for detecting the protocol inside a TLS session can be used, like those described by McCarthy [51].

The sessions using compression are removed, as only the compressed size instead of the uncompressed size can be measured for these sessions. Only a small number of sessions use compression, thus the impact of removing compressed TLS sessions is low.

As the initial request size of Exchange ActiveSync TLS traffic is significantly different from other port 443 TLS traffic, two datasets are created. One dataset including the traffic to the Exchange server and one excluding traffic to the Exchange server. Both datasets are used in the evaluation, as the presence of the Exchange traffic may significantly influence the detection results.

Summarizing, training data for the machine learning algorithms was obtained by the following steps:

1. Label and combine the initial request sizes for legitimate, tunnelled malware and native TLS malware traffic.
2. Remove all sessions on ports other than 443.
3. Remove all sessions with compression enabled.
4. Remove all sessions with an initial request size larger than 1500 bytes.
5. Optionally, remove all traffic to the university Exchange server.
6. Resample the data such that the number of malware and legitimate sessions is equal.

### 6.2.2 *Testing by using machine learning software*

The Weka machine learning toolkit [39] is used to build and evaluate the classifiers for C&C channel detection. The labelled datasets with initial request sizes are loaded as input to the different machine learning algorithms. These algorithms output a classifier which should be able to classify TLS sessions as legitimate or C&C traffic based on the initial request size. Both the detection rate and the number of false positives are evaluated using 10-fold cross-validation.

The selected classification algorithms have all been evaluated using the default values provided by Weka as these settings provide reasonable values for each algorithm. The J48 algorithm has been used both with default settings and with modified settings to reduce the size of the classification tree. The default settings include a confidence factor of 0.25 and minimum of 2 instances per leaf. The reduced tree is generated by decreasing the confidence factor to 0.1 and increasing the minimum number of instances per leaf to 50. Increasing the minimum number of items per leaf forces the algorithm to only create decisions based on a large number of samples. Decreasing the confidence factor requires the difference in number of legitimate and C&C samples to be larger before a decision node is added to the tree.

For each algorithm, the complexity of the classifier is approximated by modelling the algorithm using *detection ranges*. Those are rules of the form if $a < x < b$ then classify the session as C&C traffic, where $x$ is the size of the initial request. This complexity measure does not describe the performance of the classifier, but the complexity of the decision process captured in the classifier.

### 6.2.3 *Detection results*

An overview of the detection and false positive rate of all evaluated algorithms can be found in table 6.1 and figure 6.1. All algorithms are able to detect more than 90% of the C&C TLS sessions. While most algorithms are able to keep the number of false positives (legitimate sessions classified as C&C) below 5%. However, the lowest false positive rate is still 2.4%. Thus, detection of C&C traffic is possible using these algorithms, but the high false positive rates limit the detection capabilities in practice.

*Naïve bayes* has the worst performance of all classifiers with false positive rates of 12.0% and 44.3% for the dataset including and excluding exchange traffic respectively. Because only a single attribute is used, Naïve Bayes only models the classes using a normal distribution. Although the distributions of legitimate and C&C traffic are clearly different, these differences are not well captured using a normal distribution. This can be explained by the fact that the distribution of legitimate request sizes in figure 4.3c do not follow a normal distribution.

*Conjunctive rule* is able to achieve reasonable performance on the dataset including Exchange TLS sessions with 5.1% false positives. The resulting classifier uses only the rule $25 < x < 299$ to detect C&C traffic, where $x$ is the size of the initial request. With only this rule, it is able to detect 94.9% of C&C sessions with 5.1% false positives. In the dataset excluding Exchange TLS sessions, the false positive rate increases to 9.4%, as the initial request sizes of Exchange sessions falls within the the range of initial request sizes for C&C traffic.

*Adaboost* applied to decision stumps achieves a slightly higher false positive rate than Conjunctive rule for the dataset including Exchange traffic. However, for the dataset excluding Exchange traffic it is able to achieve a false positive rate of 3.8%, which is significantly lower than Conjunctive rule. This is achieved without increasing the complexity of the classifier as still one detection range is used.

*J48* is able to achieve good performance on both datasets using the default settings. However, performance only slightly decreases when the algorithm is limited to a much smaller tree. In the minimal case, the confidence factor was decreased from 0.25 to 0.1 and the minimum number of objects in a

| | INCLUDING EXCHANGE | | |
|---|---|---|---|
| | DETECTION RATE | FALSE POSITIVES | DETECTION RANGES |
| Adaboost | 94.9% | 5.5% | 1 |
| Conjunctive rule | 94.9% | 5.1% | 1 |
| J48 (default) | 96.1% | 3.2% | 17 |
| J48 (minimal) | 96.0% | 3.3% | 4 |
| Naïve Bayes | 97.3% | 44.3% | 1 |
| RIPPER | 95.9% | 2.9% | 7 |
| SVM | 96.0% | 2.4% | 49 |
| | EXCLUDING EXCHANGE | | |
| | DETECTION RATE | FALSE POSITIVES | DETECTION RANGES |
| Adaboost | 94.7% | 3.8% | 1 |
| Conjunctive rule | 95.1% | 9.4% | 1 |
| J48 (default) | 95.5% | 3.2% | 15 |
| J48 (minimal) | 95.5% | 3.8% | 4 |
| Naïve Bayes | 97.8% | 12.0% | 1 |
| RIPPER | 95.3% | 3.1% | 5 |
| SVM | 95.5% | 2.9% | 49 |

Table 6.1: Classification results for TLS C&C detection

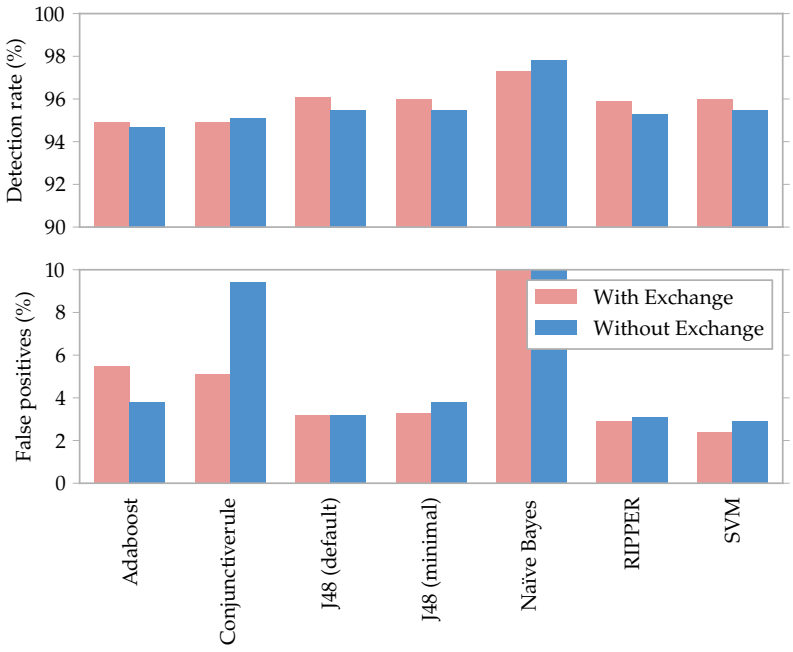



Figure 6.1: Detection results for TLS sessions

leaf increased from 2 to 50. This reduces the number of detection ranges from 17 to 4 for the dataset including exchange traffic and from 15 to 4 for the dataset excluding exchange traffic, while the false positive rate increases only slightly from 3.2% to 3.3% or from 3.2% to 3.8%. With only this slight increase, more than two thirds of the detection ranges of the default tree can be removed.

*RIPPER* has a slightly lower detection rate than J48, but more importantly it has lower false positive rates. Thus, the RIPPER detection model is likely to be preferred in practice, since false positives generate extra work for end users or administrators. To achieve these detection rates, the RIPPER detection model uses just 7 and 5 detection ranges, for the datasets including and excluding exchange traffic respectively.

*SVM* produces by far the most complex model for classification. The svm classifier model uses 49 detection ranges in order to achieve the lowest false positive rate. However, given the complexity of the model it may be overfitting on the datasets. Thus, we doubt whether these results will hold for future traffic or traffic on other networks.

### 6.2.4 *Possible improvements*

Malware uses a different initial request size for its C&C traffic than most legitimate traffic. This allows the detection technique to achieve a high detection rate. However, legitimate traffic with the same initial request size also exists, therefore the false positive rate is high. Thus, this detection technique could be improved by using extra information during the classification to reduce the number of false positives.

Focusing on a single TLS session, there are several attributes which may be used to lower the false positive rate. The signatures on the server certificate could be checked for validity. Given the analysis in section 4.4.2, it seems likely that C&C servers will use invalid certificates, whereas very few legitimate servers use invalid certificates. The size of the reply from the server can also be used. It is likely that the reply size distribution for legitimate and C&C traffic will be different, just like the request size distribution is different. Unfortunately, both attributes can not be used for the dataset containing tunnelled C&C traffic, as they are both determined by the setup used to capture TLS sessions. Thus, evaluation of these improvements would require collecting a new dataset.

Another way to reduce the number of false positives is to use this technique as part of a correlation based detection technique. Using this technique to find suspicious TLS sessions and correlating them with suspicious traffic , would improve the detection rate of correlation based techniques for TLS C&C traffic. While at the same time, keeping the low false positive rate of correlation based techniques.

### 6.2.5 *Conclusion*

Detection of TLS C&C traffic is possible by distinguishing traffic based on the size of the initial request. With the considered algorithms, around 95% of all C&C traffic can be detected. However, the best false positive rate is still 2.4%. Thus, in practice these detection methods result in too many false positives, as most traffic consists of many more legitimate TLS sessions than C&C TLS sessions. Overall, the classifier generated using RIPPER is likely

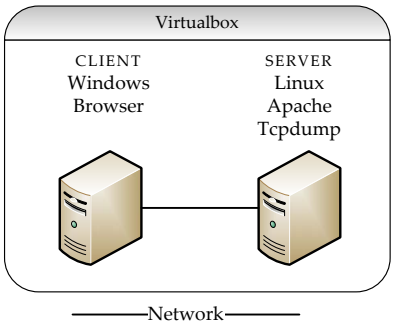



Figure 6.2: Setup for capturing browser traffic

to be the best one, providing the lowest false positive rate while avoiding possible over-fitting.

Detection results for both the datasets including and excluding Exchange traffic are similar for the best performing algorithms. However, the dataset with exchange traffic is slightly easier to classify, as a large portion of legitimate traffic has identical initial request sizes.

## 6.3 SPOOFED USER-AGENT DETECTION

The spoofed User-Agent detection method aims at detecting HTTP C&C traffic by comparing the specified User-Agent with a known fingerprint for that User-Agent. Thus, fingerprints for all common User-Agents have to be generated in order to use this method. We use these fingerprints to test the detection technique and determine both the detection rate and false positive rate.

### 6.3.1 *Building a model of legitimate browsers*

The detection method requires fingerprints for every major browser to be available. Fingerprint generation needs to take into account the subtle differences between HTTP implementations. Thus, it is very important to know with certainty which browser generates a request. Therefore, the browsers are installed on a clean virtual machine and used to browse a local website. We have taken care to ensure that the local website contains a variety of different elements such that different request generation code may be exercised.

The setup used for collecting HTTP requests is shown in figure 6.2. It consists of a *client machine* and a *server machine*. The *client machine* runs Windows and has the browser installed. The *server machine* runs linux and hosts a website using Apache. All HTTP requests are captured using tcpdump on the server.

The following browsers have been installed on the client machine for fingerprint generation:

- Internet Explorer 5.01, 6.0, 7.0, 8.0 & 9.0
- Firefox 1.0, 1.5, 2.0, 3.0 & 3.5
- Opera 9.0, 9.5, 10.0, 10.5 & 11.0
- Safari 3.1, 4.0 & 5.0

- Chrome 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 & 12
- Maxthon 2.0 & 2.5

All browsers were installed only under Windows XP SP3, with the exception of Internet Explorer. Internet Explorer and Windows are closely integrated with each other. Thus, Internet Explorer may behave differently on different versions of Windows. Therefore, each version of Internet Explorer has also been installed under the supported versions of Windows 2000, XP, Vista and 7. The different installations of all browsers have been used to browse the local website and the generated traffic has been collected.

We have grouped the collected traffic per major browser version. For each major browser version, we have generated a DFA model of the possible header orders. The precise steps taken to generate the fingerprints are described in section 5.2.1.

While generating the fingerprints, it became clear that Internet Explorer 5.0 has two clearly separate paths for generating HTTP requests. The first path generates all requests for webpages, images, stylesheets, etc. while the second path generates all XMLHTTPRequests, which are, for example, used by Ajax websites. This second path is implemented as a separate ActiveX component instead of being part of the browser. As two separate request generation paths are used, two different graphs have been generated for IE 5. A request with the IE5 User-Agent is considered legitimate if it matches either graph.

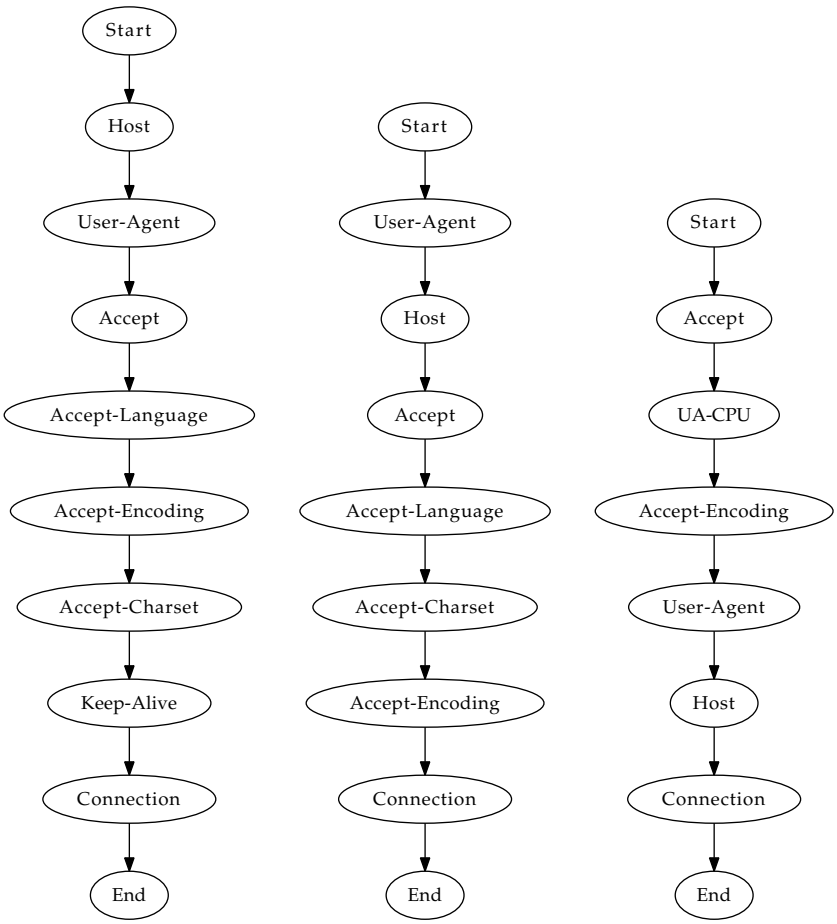
A quick check of the HTTP requests generated on operating systems other than Windows indicates that Firefox and Opera use the same code to generate requests on each platform. Safari, however, uses a different HTTP implementation on Mac OS X than on Windows. Therefore, the Safari fingerprints are marked as Safari on Windows fingerprints instead of just Safari fingerprints.

### 6.3.2 *Fingerprint discussion*

A selection of generated fingerprint graphs is included in figure 6.3 and discussed in this section. The full set of fingerprints can be found in appendix B. Most fingerprints are very simple: they consist of only one path through the graph. Thus, the requests generated by these clients always contain the same set of headers in the same order. This is, for example, the case for Firefox (6.3a) which generates requests containing the same 8 headers in a fixed order. The fingerprint for Opera (6.3b) looks similar, but has a different order for several headers and contains one header less. For both Opera and Firefox, the fingerprints for all major versions installed look identical. Thus, it is likely that the HTTP request generation code has not changed significantly between these versions.

While most browser's fingerprints are very simple, a few are more complex. The fingerprint for Safari 5 on windows (6.3e) is the most complex fingerprint obtained. Although it contains only 6 headers, it allows very different header orders.

Comparing the fingerprints for IE 7 (6.3c) and IE 8 & 9 (6.3d) it is clear that the HTTP request generation code changed between these versions. The Accept-Encoding and User-Agent header are sometimes swapped in IE 8 & 9, while IE 7 always uses the same order. Furthermore, the UA-CPU header present in IE 7 is not present any-more in IE 8 & 9.




(a) Firefox 1,2 & 3 (b) Opera 9,10 & 11 (c) IE 7

(d) IE 8 & 9 (e) Safari 5 on Windows

Figure 6.3: Header order models of different browsers

| USER AGENT | REQUESTS | | | SAMPLES | | |
|---|---|---|---|---|---|---|
| | TOTAL | DETECTED | | TOTAL | DETECTED | |
| IE 5 | 14,979 | 14,979 | 100.0% | 55 | 55 | 100.0% |
| IE 6 | 15,484 | 15,313 | 98.9% | 6,521 | 6,511 | 99.8% |
| IE 7 | 106 | 106 | 100.0% | 8 | 8 | 100.0% |
| Maxthon 2 | 3 | 0 | 0.0% | 1 | 0 | 0.0% |
| Opera 9 | 1 | 1 | 100.0% | 1 | 1 | 100.0% |
| Opera 10 | 5 | 5 | 100.0% | 1 | 1 | 100.0% |
| No fingerprint available | 238 | 0 | 0.0% | 81 | 0 | 0.0% |
| No User-Agent | 1,381 | 0 | 0.0% | 83 | 0 | 0.0% |
| Total | 32,197 | 30,404 | 94.4% | 6,751 | 6,576 | 97.4% |

Table 6.2: Malware detection results for DFA method

### 6.3.3 *Detection results*

The detection rate was evaluated using the HTTP C&C requests captured using the setup described in section 4.1.2. This dataset of malware HTTP requests contains a total of 32197 requests generated by 6576 samples. The number of HTTP requests is much larger than the number of samples. This is caused in part by the fact that all malicious documents were opened in multiple versions of the viewing application, thus malware may have been deployed multiple times. Another reason is that a small set of malware generates hundred of requests per minute, with one sample sending more than 800 http requests in one minute.

All malware C&C requests are parsed to obtain the claimed User-Agent and the header order. The correct fingerprint for the claimed User-Agent is retrieved and checked against the observed header order. If the observed header order can not be generated using the fingerprint, the request is marked as C&C request. During this process all headers which are not in the fingerprint are ignored as headers have been removed during fingerprint generation as well.

As can be seen in table 6.2, nearly all HTTP requests generated by malware are detected. Looking at the raw numbers, 94.4% of all requests generated by malware are detected. For almost all of the samples which are not detected, the User-Agent is unrecognised. These samples are ignored as there is no fingerprint available for comparison. The detection rate when requests are grouped per sample is even better, with 97.4% of malware detected. Again, almost all malware samples that are not detected use unknown User-Agents. Only 11 samples for which a fingerprint is available are not detected.

As indicated in section 4.3.4, most malware for which the Internet Explorer 6 User-Agent was observed, always uses the User-Agent of the installed Internet Explorer version. Thus, the detection rate may differ depending on the installed version of Internet Explorer. Therefore, all requests using the Internet Explorer 6 User-Agent were also evaluated using fingerprints of other Internet Explorer versions. As can be seen in table 6.3, the detection rates are high for Internet Explorer 5 through 7. However, most

| SIGNATURE | REQUESTS DETECTED | | SAMPLES DETECTED | |
|---|---|---|---|---|
| IE 5 | 15313 | 98.9% | 6511 | 99.8% |
| IE 6 | 15313 | 98.9% | 6511 | 99.8% |
| IE 7 | 15484 | 100.0% | 6521 | 100.0% |
| IE 8/9 | 230 | 1.5% | 98 | 1.5% |

Table 6.3: Malware detection results when using different IE fingerprints

C&C traffic matches the Internet Explorer 8 & 9 fingerprints. This is most likely caused by the small number of headers in the IE 8&9 fingerprint (figure 6.3d), especially given that the Host header is required by the HTTP 1.1 specification and the User-Agent header is required for spoofed User-Agent detection. Thus, unfortunately, when running the latest versions of Internet Explorer the detection rates are very low with 2.4% of the samples detected, compared to 97.4% or higher for older versions.

A detection method for C&C traffic should not only have a high true positive rate, but must also have a low false positive rate, to be effective. Investigating detected sessions becomes too much work if many false positives are generated, while automatically blocking detected sessions will hinder users too much.

The false positive rate was evaluated using the dataset described in section 4.3.1. The detection results on the legitimate requests are presented in table 6.4a. It is assumed that the dataset does not contain C&C traffic. Therefore, it should report a very low number of detections. As can be seen in table 6.4a this is not the case. For example, 72.3% of all Internet Explorer 6 requests are considered anomalous. A sample of the detected requests were examined and if an obvious explanation could be found for the requests a filter was written to remove these requests.

The filters obtained can be found in table 6.4b. The most important filter to reduce the false positive rate is to filter out Lo-Jack laptop tracking software. The laptop tracking software regularly makes requests to "search.namequery.com" with an User-Agent identifying itself as Internet Explorer 6.0. These requests are anomalous as the software is spoofing the User-Agent, they are however not C&C requests and are therefore filtered from the dataset.

A filter was also added for an User-Agent containing a combination of Firefox version and Gecko[1] version which was never released. The User-Agent in these requests was clearly spoofed, but given the request contents they look legitimate and are therefore filtered.

Three filters were added for requests in which the Accept-Encoding was masked out by replacing each character with the – or X character. According to Gentilcore [33], such requests are modified by desktop security software before transmission. Thus, these requests are generated by the specified User-Agent, but do not match the fingerprint because they have been modified. These requests have been filtered, as they are false positives.

After filtering, the false positive rate has been reduced to 0.5% of all filtered requests for which a fingerprint is available. The great majority of the requests detected still look legitimate upon quick inspection. However, upon inspection of the anomalous Internet Explorer 5 requests, it quickly becomes clear that none of the requests are actually generated by Internet Explorer 5.

1 Gecko is the rendering engine used by Firefox

| USER AGENT | BEFORE FILTERING | | | AFTER FILTERING | | |
|---|---|---|---|---|---|---|
| | TOTAL | DETECTED | | TOTAL | DETECTED | |
| Chrome 2 | 23,925 | 595 | 2.5% | 23,330 | 0 | 0.0% |
| Chrome 3 | 5,930 | 0 | 0.0% | 5,930 | 0 | 0.0% |
| Chrome 4 | 17 | 17 | 100.0% | 17 | 17 | 100.0% |
| Firefox 1 | 8 | 8 | 100.0% | 8 | 8 | 100.0% |
| Firefox 2 | 15,711 | 2,894 | 18.4% | 13,002 | 185 | 1.4% |
| Firefox 3 | 306,849 | 2,030 | 0.7% | 305,355 | 536 | 0.2% |
| IE 5 | 285 | 285 | 100.0% | 145 | 145 | 100.0% |
| IE 6 | 20,068 | 14,514 | 72.3% | 5,886 | 332 | 5.6% |
| IE 7 | 27,188 | 857 | 3.2% | 26,999 | 668 | 2.5% |
| IE 8 | 57,555 | 759 | 1.3% | 57,247 | 451 | 0.8% |
| Maxthon 2 | 48,178 | 146 | 0.3% | 48,032 | 0 | 0.0% |
| Opera 9 | 24,074 | 3,086 | 12.8% | 21,127 | 139 | 0.7% |
| Opera 10 | 11,497 | 35 | 0.3% | 11,477 | 15 | 0.1% |
| Safari 3 | 42 | 42 | 100.0% | 41 | 41 | 100.0% |
| Safari 4 | 205 | 0 | 0.0% | 205 | 0 | 0.0% |
| Seamonkey 1 | 22,654 | 0 | 0.0% | 22,654 | 0 | 0.0% |
| Total (fingerprint available) | 564,186 | 25,268 | 4.5% | 541,455 | 2,537 | 0.5% |

(a) False positives

| NR. OF REQUESTS | FILTER |
|---|---|
| 14,170 | Request made by Lo-Jack laptop tracking software |
| 2,934 | Opera Accept-Encoding header replaced by X's |
| 2,709 | Requests with impossible Gecko version and Firefox version combination |
| 1,258 | Firefox Accept-Encoding headers replaced by –'s |
| 593 | Chrome Accept-Encoding headers replaced by –'s |
| 314 | Office Live suite sending requests as Internet Explorer |
| 203 | Windows proxy auto configuration request |
| 167 | TCP stream reassembly problem |
| 150 | Firefox wipmania plugin |
| 140 | Internet Explorer 5 on Mac OS X |
| 93 | Alexa Toolbar for Internet Explorer |

(b) Filtering rules

| | NR. OF REQUESTS | |
|---|---|---|
| Fingerprint available | 564,186 | 77.2% |
| Fingerprint not available | 162,152 | 22.2% |
| User-Agent not in request | 4,172 | 0.6% |

(c) Fingerprint availability

Table 6.4: False positives for spoofed User-Agent detection

| USER-AGENT | TOTAL | DETECTED | |
|---|---|---|---|
| Chrome 8 | 1,520 | 0 | 0.0% |
| Chrome 10 | 3,214 | 0 | 0.0% |
| Chrome 11 | 6,147 | 0 | 0.0% |
| Chrome 12 | 751 | 0 | 0.0% |
| Firefox 3 | 44,320 | 3 | 0.0% |
| IE 5 | 8,494 | 8,494 | 100.0% |
| IE 6 | 141,626 | 3,554 | 2.5% |
| IE 7 | 4,937,065 | 3,318 | 0.1% |
| IE 8 | 1,822,380 | 35 | 0.0% |
| IE 9 | 34 | 1 | 2.9% |
| Opera 11 | 1 | 1 | 100.0% |
| Total (fingerprint available) | 6,965,552 | 15,406 | 0.2% |

(a) False positives

| | NR. OF REQUESTS | |
|---|---|---|
| Fingerprint available | 6,965,552 | 80.9% |
| Fingerprint not available | 1,643,251 | 19.1% |
| User-Agent not in request | 4,862 | 0.1% |

(b) Fingerprint availability

Table 6.5: False positives for spoofed User-Agent detection on 2nd dataset

We have also estimated the false positive rate using a second dataset containing web traffic obtained from a company network. This dataset is more recent and thus contains different browser versions. Furthermore, measuring the false positive rate on a different network may provide better insight into the false positive rates on different networks.

The dataset contains requests sent via a proxy, this proxy re-orders the Connection, Host and Keep-Alive headers in the request. Therefore, these headers have been removed from the fingerprints before evaluating the detection method. The detection results on the dataset can be found in table 6.5a. The false positive rate on this dataset is only 0.2%. However, the Internet Explorer 5 requests were generated by antivirus software, which uses an Internet Explorer 5 User-Agent. Ignoring these requests, the false positive rate is only 0.1%.

### 6.3.4 *Possible improvements*

Using spoofed User-Agent detection for C&C channel detection works well for most User-Agents. Unfortunately, most malware spoofs the User-agent of the installed Internet Explorer version and the fingerprint of Internet Explorer 8 & 9 (figure 6.3d) is so simple that most C&C traffic matches it.

Thus, the current detection algorithm is not able to detect most C&C traffic if Internet Explorer 8 or 9 is installed on the infected computer.

The current fingerprints are highly generalized, thus it may be possible to improve the detection rate rate using a more detailed fingerprint of Internet Explorer 8 & 9. There are several ways to create a more detailed fingerprint. Headers which are only present for a subset of requests may be included. Multiple fingerprints may be created for different types of requests (e.g. html pages, images, XMLHTTPRequests) Another possibility would be to include certain header values in the fingerprint. The Accept-Encoding header could, for example, be used as it contains a fixed value for a specific browser version, but may be different between browser versions.

A large selection of requests with spoofed User-Agents are actually generated by legitimate software, leading to many false positives. The number of false positives may be reduced by including filters or signatures for legitimate software known to spoof the User-Agent. Unfortunately, this also increases the work needed for generating the signatures.

Another possibility to improve spoofed User-Agent detection is to combine information about multiple requests. The advantage of combining multiple requests is that single false positives may be suppressed based on the rest of the requests. The requests may, for example, be combined based on their TCP stream or their referrer header. These grouped requests may be considered legitimate if, for example, at least 90% of the requests match the fingerprint.

The false positive rate may also be reduced by combining the spoofed User-Agent detection with, for example, a domain reputation score (e.g. NOTOS [15]). Combining these systems, a lower false positive rate may be achieved while keeping the high detection rate.

### 6.3.5 *Conclusion*

Detecting C&C traffic using spoofed User-Agents seems to work well, using our test setup. The spoofed User-Agent detection technique is able to detect 97.4% of the samples tested. However, if the malware would have been executed on a machine running Internet Explorer 8 or 9, the detection rate would have been much lower. An exact detection rate can not be given since not all malware was run on such a machine, but the detection rate is estimated to be around 6%. (Based on 4% of the Internet Explorer 6 User-Agents being hard coded as estimated in section 4.3.4.)

Even after manually adding filters, the false positive rate of 0.5% on the first dataset is still too high for many practical uses. The false positive rate of 0.1%, with just one filter, on the second dataset is much better. However, one false positive for every thousand requests is still too high for most uses. One of the reasons for the high false positive rate is the significant set of legitimate programs providing an incorrect User-Agent. To reduce false positives, fingerprints or filters have to be created for these applications. This adds complexity to the filter and may decrease the sensitivity of the detection.

## 6.4 $2_v$-GRAM BASED ANOMALY DETECTION

The $2_v$-gram based anomaly detection method has to be trained using legitimate traffic in order to build a model of legitimate traffic. The trained model can then be used for detection of anomalous requests which should include C&C requests.

| PARAMETER | VALUES | DESCRIPTION |
|---|---|---|
| $n$ | 5, 10 | $2_v$-grams for $v = 0..n-2$ |
| $k$ | 10, 20, 40, 80, 160 | Nr. of clusters for $2_v$-grams |
| $\gamma$ | 0.5 | Width of radial base function kernel |
| $f_{combination}$ | Average, Minimum | Function to combine SVM scores |
| $fp_{desired}$ | 0.5%, 0.25% | Desired false positive rate |

Table 6.6: Parameters for $2_v$-gram evaluation

### 6.4.1 *Training*

We have trained the $2_v$-gram based anomaly detection system using the legitimate HTTP requests in the dataset described in section 4.3.1. For this purpose we have extracted all HTTP requests in the dataset without their message body (e.g. POST data). The message bodies are ignored because they can significantly influence the $2_v$-gram distribution, while 99% of the captured C&C requests do not contain message bodies. All duplicate requests are also removed and the false positive rate is evaluated using 10-fold cross-validation. The detection rate is evaluated with the C&C HTTP requests captured using the setup described in section 4.1.2. The entire dataset has been labelled using each of the 10 detectors trained for cross-validation. The average of the detection rate among these 10 systems is taken for evaluation.

For evaluation of the detection method 7-grams and 12-grams are approximated with $v = 0..5$ and $v = 0..10$. The number of clusters evaluated ranges from 10 to 160 clusters by setting `clusters` to 10, 20, 40, 80, and 160. After some initial tests ,$\gamma = 0.5$ was chosen as width of the radial base function as it seems to provide good results for classification. The scores are combined by either taking the average or the minimum as combination function $f_{combination}$. The desired false positive rates $fp_{desired}$ are set at 0.5% and 0.25%.

### 6.4.2 *Detection results*

The achieved detection rates for the different parameters can be found in table 6.7. The results show that the detection rate is insensitive to different values of the selected parameters. Almost every combination of parameters achieves a 29% detection rate. As expected, the false positive rate is mostly influenced by $fp_{desired}$. However, it is interesting to note that the achieved false positive rate is significantly smaller than $fp_{desired}$ when it is set to 0.5% in combination with a small number of clusters. In the best case, false positive rates of 0.15% in combination with detection rates of 29.43% are possible.

### 6.4.3 *Possible improvements*

The achieved detection rate of $2_v$-gram anomaly detection is relatively low, with only 29%. The most likely reason for this is the similarity between legitimate and C&C HTTP requests. Both requests use the same keywords and are probably too similar when looking at the $2_v$-grams. Thus, better detection results may be obtained by transforming the HTTP requests. Common

| | | detection rate | | | |
|---|---|---|---|---|---|
| | | $fp_{desired} = 0.5\%$ | | $fp_{desired} = 0.25\%$ | |
| | k | $\nu = 0..5$ | $\nu = 0..10$ | $\nu = 0..5$ | $\nu = 0..10$ |
| average probability | 10 | 29.40% | 29.38% | 29.27% | 14.04% |
| | 20 | 29.39% | 29.38% | 29.38% | 18.93% |
| | 40 | 29.39% | 29.38% | 29.38% | 23.30% |
| | 80 | 29.38% | 29.38% | 29.38% | 16.97% |
| | 160 | 29.38% | 29.38% | 0.15% | 2.36% |
| minimum probability | 10 | 29.53% | 29.45% | 7.64% | 19.68% |
| | 20 | 29.49% | 29.44% | 29.38% | 25.30% |
| | 40 | 29.52% | 29.46% | 29.21% | 22.51% |
| | 80 | 29.20% | 24.63% | 29.21% | 21.76% |
| | 160 | 23.68% | 26.93% | 23.99% | 8.74% |
| | | achieved false positive rate | | | |
| | | $fp_{desired} = 0.5\%$ | | $fp_{desired} = 0.25\%$ | |
| | k | $\nu = 0..5$ | $\nu = 0..10$ | $\nu = 0..5$ | $\nu = 0..10$ |
| average probability | 10 | 0.52% | 0.51% | 0.20% | 0.24% |
| | 20 | 0.50% | 0.53% | 0.24% | 0.25% |
| | 40 | 0.50% | 0.50% | 0.25% | 0.24% |
| | 80 | 0.51% | 0.50% | 0.24% | 0.23% |
| | 160 | 0.55% | 0.52% | 0.28% | 0.26% |
| minimum probability | 10 | 0.32% | 0.35% | 0.19% | 0.13% |
| | 20 | 0.29% | 0.31% | 0.15% | 0.21% |
| | 40 | 0.33% | 0.34% | 0.14% | 0.14% |
| | 80 | 0.25% | 0.28% | 0.14% | 0.13% |
| | 160 | 0.55% | 0.50% | 0.29% | 0.24% |

Table 6.7: Detection and false positive rates for $2_\nu$-gram detection

keywords can, for example, be tokenized to enhance the difference between legitimate and C&C traffic. Another approach would be, to only extract values which are likely to be discriminatory, like the URL, host, cookies and unknown headers and use these for anomaly detection.

Another approach would be to build multiple $2_v$-gram detection models for specific types of requests. This would allow the detection models to be more specific and thus more likely to detect C&C traffic as an anomaly. However, it would require a good classification method for determining the type of request.

The results of $2_v$-gram anomaly detection can also be combined with another system using, for example, correlation based techniques. This would provide the ability to maintain or lower the false positive rate, while increasing the detection rate.

### 6.4.4 *Conclusion*

The $2_v$-gram anomaly detection method is not able to detect most malware. However, low false positive rates of 0.15% can be achieved. This might make this method suitable in a system which combines several independent detection methods. The combined system may be able to achieve higher detection rates while maintaining the low false positive rate.

# 7 CONCLUSION

We have created and analysed a dataset containing web based C&C traffic. Based on this analysis we have proposed and evaluated three detection techniques for web based C&C channels. We have shown that each technique is capable of detecting C&C channels, however their current false positive rates are still high. Thus, future research may focus on improving these techniques to reduce the false positive rate.

In the rest of this chapter we will revisit the research sub-questions and the main research question. Finally, we conclude with an overview of research directions which can be taken to improve the proposed detection techniques.

## 7.1 ANSWERING THE RESEARCH SUB-QUESTIONS

*How can a dataset of C&C web traffic be obtained?*

In chapter 4 we have shown that a dataset containing C&C web traffic can be obtained by running malware in a controlled environment. The malware needed to create the dataset is freely available on the Internet. While a controlled environment can be easily constructed using virtual machines. The disadvantage of a controlled environment is that it limits the C&C traffic which can be collected, because C&C server software is often not available.

*How prevalent is the usage of HTTP based C&C channels in malware? What are distinguishing characteristics of HTTP C&C traffic?*

The majority of the samples analysed in chapter 4 use HTTP based C&C channels. Similar observations have been found in literature confirming that HTTP is the dominant C&C protocol. A precise estimate of the usage of HTTP based C&C channels can not be made, but given the results we can be fairly certain that more than half of all C&C channels are currently HTTP based.

The most distinguishing characteristic of HTTP based C&C channels is that the User-Agent in the requests is spoofed. The majority of malware uses the User-Agent of the installed Internet Explorer version in its requests. Another characteristic is that on average C&C HTTP requests are smaller than legitimate requests.

*How prevalent is the usage of C&C channels on port 443 in malware? How prevalent are TLS or SSL C&C channels in malware?*

Of the samples analysed in chapter 4, half of the malware families and 6.0% of the malicious office documents generates traffic on port 443. However, only 0.6% of the PDF documents generated traffic on port 443. These number most likely overestimate the usage of port 443 in malware, as the samples were collected with the focus on obtaining malware using TLS or port 443. Given the low prevalence, even for samples collected with a focus on port 443 and TLS usage, the actual prevalence is likely to be extremely low.

TLS C&C channels are rarely used by malware. While specifically searching for malware using TLS C&C channels only four malware families and one malicious document were found to use it. A further fifth malware family was found to support it, but no samples could be found in the wild using this feature. Thus, while TLS support could be easily added to current malware it is rarely used at the moment.

*Which method works best to distinguish legitimate TLS or SSL traffic from TLS or SSL C&C channels? What detection and false positive rates can be achieved?*

Too few samples are available to design or evaluate a general detection technique for TLS C&C channels. Instead, we have proposed and evaluated a detection technique by simulating the effect of adding TLS support to current malware. This was done by tunnelling HTTP based C&C traffic over a TLS channel. Given the limitations of tunnelling traffic described in section 4.2.2, we show that C&C traffic can be distinguished from legitimate traffic by observing the size of the initial HTTP request send over the TLS channel.

Several methods were evaluated for distinguishing legitimate TLS traffic from TLS C&C channels. The best detection and false positive rates were achieved using SVM classification. This allowed for a 96.0% detection rate combined with a 2.4% false positive rate. However, these rates may have been the result of over-fitting on the evaluation datasets. Thus, we consider the results of the RIPPER algorithm more realistic with a detection rate of 95.9% and a false positive rate of 2.9%.

*Which method works best to distinguish legitimate HTTP traffic from C&C HTTP traffic? What detection and false positive rates can be achieved?*

We have proposed two techniques for distinguishing legitimate web traffic from C&C traffic in chapter 5. The first method is based on detecting spoofed User-Agents, as the majority of malware spoofs the User-Agent for their C&C HTTP traffic. The second method is based on detecting anomalous HTTP requests based on the $2_v$-gram distribution of the request.

The spoofed User-Agent method is able to achieve a false positive rate of 0.5% in combination with a few manual filters. While the anomaly detection method is able to achieve a 0.14% false positive rate. However, their detection rates differ significantly. The spoofed User-Agent method achieves a detection rate of 97.4%, which is much higher than the 29.2% detection rate achieved by the anomaly detection method. Thus, high detection rates can be achieved. However, the false positive rates remain too high for many practical uses. On a large network there are likely to be multiple false positives every minute. Thus, making it impractical to look at each detection manually or block all detected requests.

*How do we set-up proper experiments to measure both the "detection rate" and the "false positive rate" of each technique?*

We have proposed an experiment to measure both the "detection rate" and the "false positive rate" of each technique in section 6.1. The detection rate is estimated by measuring the detection rate on a dataset containing C&C traffic. This dataset has been collected in a controlled environment as described in sections 4.1 and 4.2. The false positive rate is estimated by measuring the detection rate on network traffic collected at a university network. This dataset is described in section 4.3.1. The assumption is made that the

university network contains only legitimate traffic, therefore the detection rate on this dataset is taken as an estimate for the "false positive rate".

To make sure that the detection and false positive rates are measured on a different dataset than used for training, 10-fold cross-validation is used. This splits up the dataset to ensure that all elements can be evaluated using a detector trained on other elements. Cross-validation also helps to ensure that the results remain stable when re-training. In case training is not required for the detection techniques, the entire dataset is used for evaluation.

## 7.2 ANSWERING THE RESEARCH QUESTION

*How can we distinguish C&C web traffic from legitimate web traffic in both the encrypted and unencrypted case?*

We have shown that encrypted C&C web traffic is rare, but if encryption would be added to current unencrypted C&C web traffic, detection is possible by looking at the size of the HTTP request. However, using just this measure the false positive rate is too high for most uses.

We have shown that web based C&C channels can be distinguished from legitimate traffic because the User-Agent for C&C channels is usually spoofed. Detection of C&C channels based on spoofed User-Agent detection provides good detection results for most User-Agents. However, it needs to be combined with filters for legitimate applications which also spoof their User-Agent.

Using anomaly detection techniques to distinguish legitimate traffic from C&C traffic is also possible. However, in the implementation we have evaluated the detection rates are relatively low.

## 7.3 FUTURE RESEARCH

We have shown that legitimate web traffic can be distinguished from C&C web traffic. However, the detection and false positive rates of the techniques we have proposed, have to be improved before they can be widely used. In chapter 6, we have mentioned several directions of research to improve the detection techniques.

### 7.3.1 *Machine learning-based TLS classification*

Detection based on just the initial request size provides good detection rates. However, using only one variable results in high false positive rates. Thus, future research may focus on finding other distinguishing attributes for TLS C&C traffic and incorporating those in the detection system. Such attributes may, for example, be the server reply size, certificate or domain name.

Another direction of research would be to incorporate the TLS C&C channel detection into a correlation based system. The correlation based system may take suspicious TLS sessions as input, but can suppress false positives by correlating these sessions with other suspicious events.

### 7.3.2 *Spoofed User-Agent detection*

Detecting C&C channels using spoofed User-Agent detection works very well for most User-Agents. However, the proposed algorithm does not work

very well for detecting C&C channels using the Internet Explorer 8 or 9 User-Agent. Thus, future research may focus on creating a better fingerprint such that these C&C channels can be detected as well.

Reducing the false positive rate might be possible by taking into account more information. Future research could, for example, combine multiple requests or take into account both the request and DNS traffic.

### 7.3.3 $2_v$-*gram based anomaly detection*

Future research may focus on how to improve the detection rate of the $2_v$-gram based anomaly detection technique. One method to increase the difference in $2_v$-gram distribution between legitimate and C&C traffic would be to replace common keywords with tokens. Another possibility would be to classify requests into different classes and build a more specific $2_v$-gram detector for each class.

# A

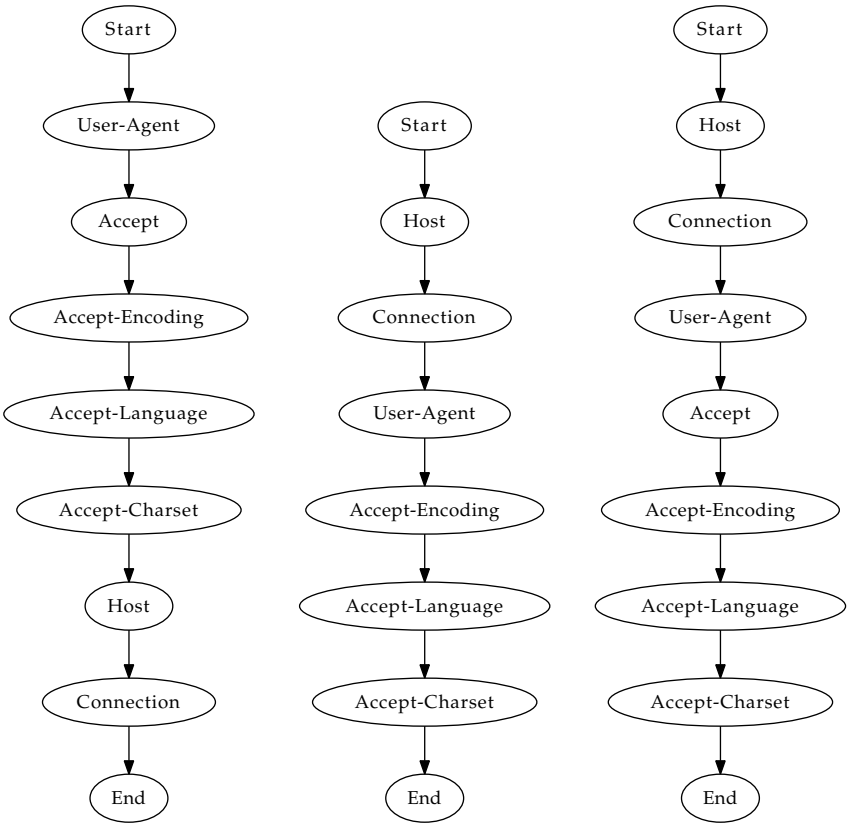
## OVERVIEW OF MALWARE FAMILIES ANALYSED

| FAMILY NAME | DESCRIPTION OF TRAFFIC AS FOUND ON THE INTERNET |
|---|---|
| *Agobot* | "uses Secure Socket Layer (SSL)"<br>http://about-threats.trendmicro.com/ArchiveMalware.aspx?language=us&name=WORM_AGOBOT.JT |
| *Ghegbot (Mondera. Tofsee)* | "Uses port 443 (SSL) to send and receive encrypted commands. spam templates and download executable files."<br>http://www.m86security.com/labs/spambotitem.asp?article=897 |
| *Hydraq* | "Establishing an encrypted covert channel that masqueraded as an SSL connection"<br>http://www.wired.com/threatlevel/2010/01/operation-aurora |
| *Mebroot (Sinowal. Mebload)* | "Some of the websites it is known to connect to are ... via TCP port 443"<br>http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=TrojanDownloader:Win32/Sinowal.A |
| *Mega-D* | "Reports to control server on port 80. also uses encrypted HTTPS on port 443"<br>http://www.m86security.com/labs/spambotitem.asp?article=896 |
| *NTESSESS* | "communicates ... over TCP port 443. and actually uses SSL"<br>http://www.cyberesi.com/2011/05/12/malware-obfuscated-within-png-files-sample-2-2 |
| *PingBed* | "This Trojan uses SSL for its network communication"<br>http://www.cyberesi.com/2011/05/10/malware-obfuscated-within-png-files |
| *Pushdo (Cutwail. Pandex)* | "encrypted its communications and routed them through the SSL port (443); while this encryption looked like SSL at first sight" ..." it is actually NOT." "There is a routine which generates some actual SSL traffic to a list of 339 known web sites"<br>http://blog.fortinet.com/pushdo-revolutions-communication-encryption-and-decoy-traffic |

| FAMILY NAME | DESCRIPTION OF TRAFFIC AS FOUND ON THE INTERNET |
|---|---|
| *Ramnit.C* | "The connection to a server which W32/Ramnit.C initiates uses TCP port 443."<br>http://techblog.avira.com/2010/11/25/closer-look-at-w32ramnit-c/en |
| *Routrobot (buzus. prolaco)* | "Connects to the following IP Addresses ... through remote port 443."<br>http://www.mcafee.com/threat-intelligence/malware/default.aspx?id=256356 |
| *Rustock* | "the Rustock botnet has been sending a lot more spam using TLS"<br>http://www.symantec.com/connect/blogs/death-thousand-cuts-rustock-botnet-sending-more-encrypted-spam |
| *Spybot.worm .gen.p* | "opens a backdoor at TCP port 443 and tries to connect to IRC server"<br>http://www.mcafee.com/threat-intelligence/malware/default.aspx?id=135336 |
| *Spyeye* | "connects to ... through a remote port 443"<br>http://vil.nai.com/vil/content/v_494378.htm |
| *Swrort* | "is known to connect to the following servers: ... via TCP port 443"<br>http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Trojan:Win32/Swrort.A |
| *Tdss* | "data transmitted to and from C&C over HTTP/HTTPS"<br>http://resources.infosecinstitute.com/tdss4-part-2 |
| *Zeus (Zbot)* | Implements support for HTTPS C&C (see source code analysis in section 4.4.1) |

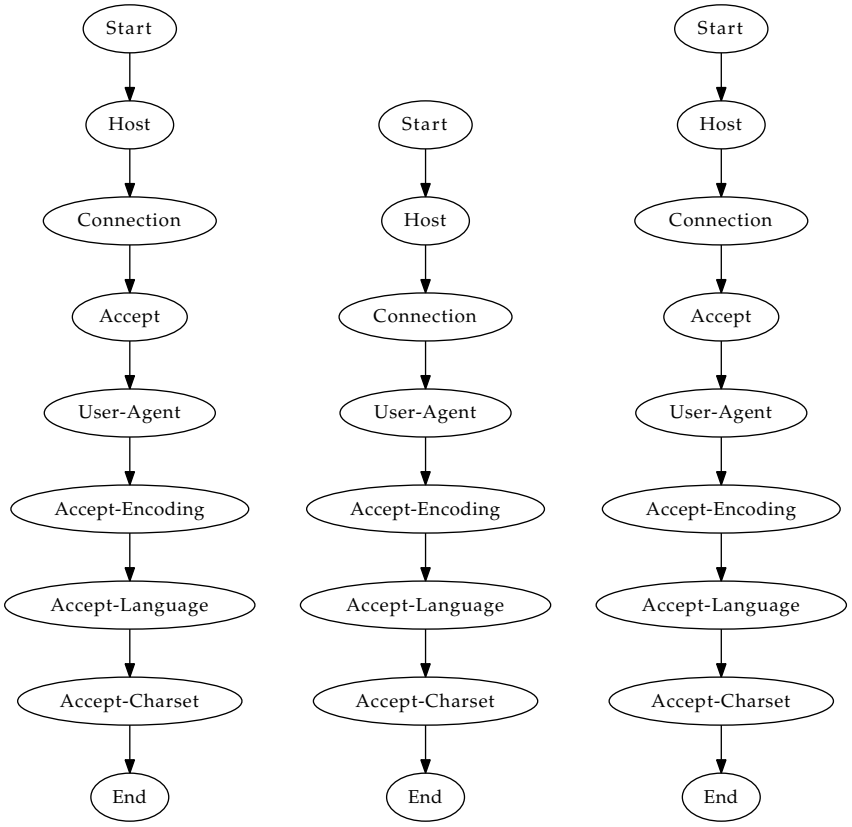
# B

## BROWSER FINGERPRINTS

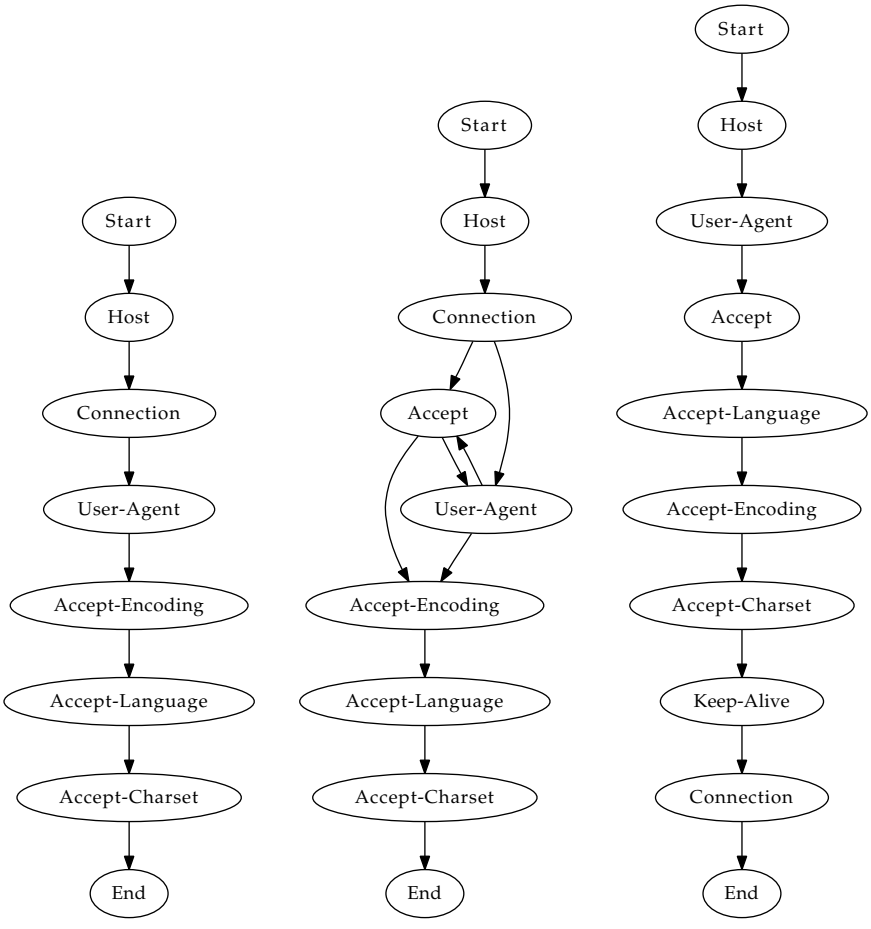



(a) Chrome 1 (b) Chrome 2&3 (c) Chrome 4&5

Figure B.1: Chrome fingerprints

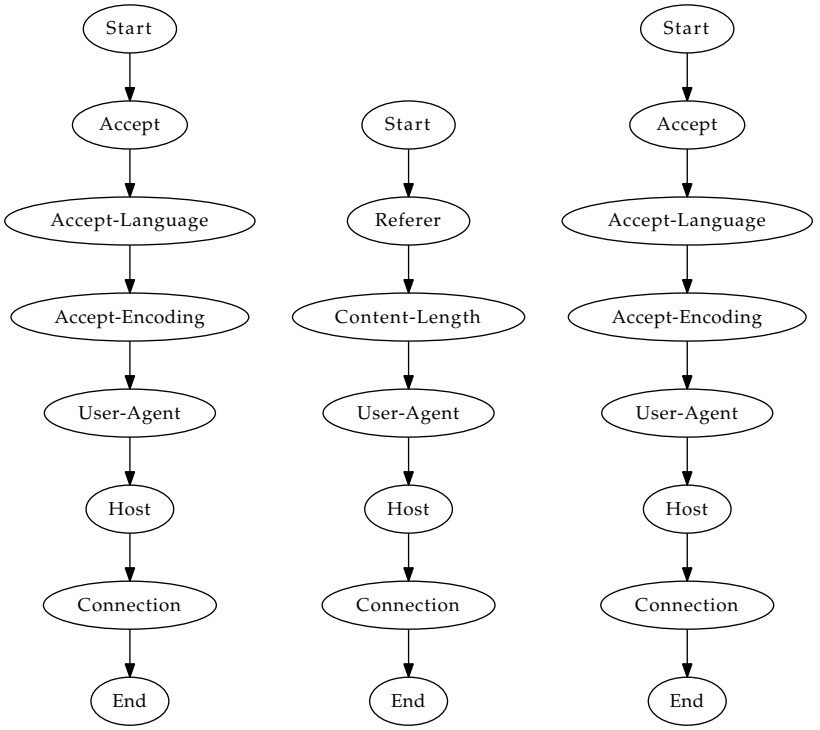



(a) Chrome 6 (b) Chrome 7&8 (c) Chrome 9

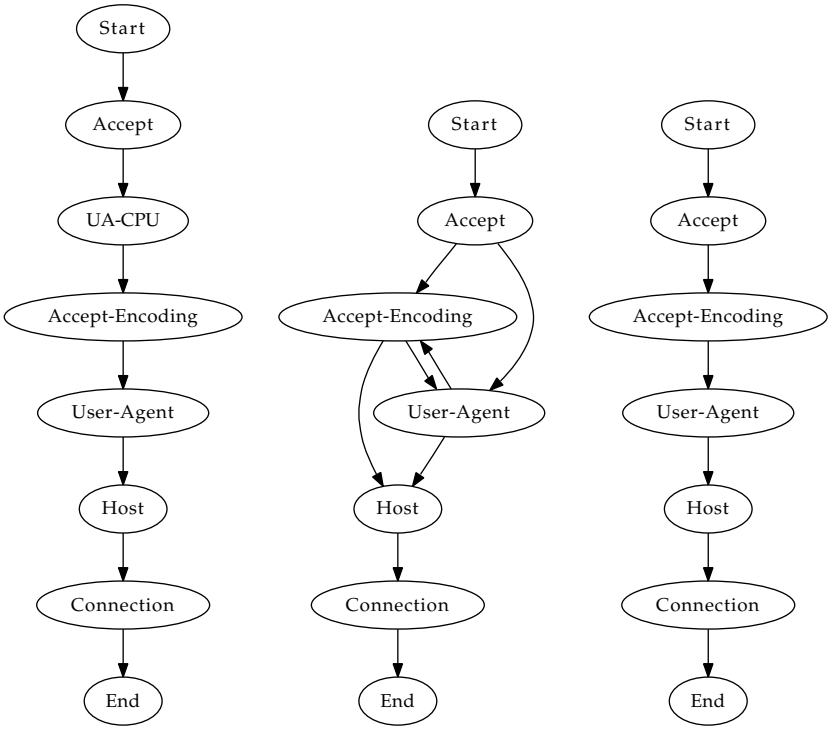



(d) Chrome 10&11 (e) Chrome 12 (f) Firefox 1,2 & 3

Figure B.2: Chrome & Firefox fingerprints

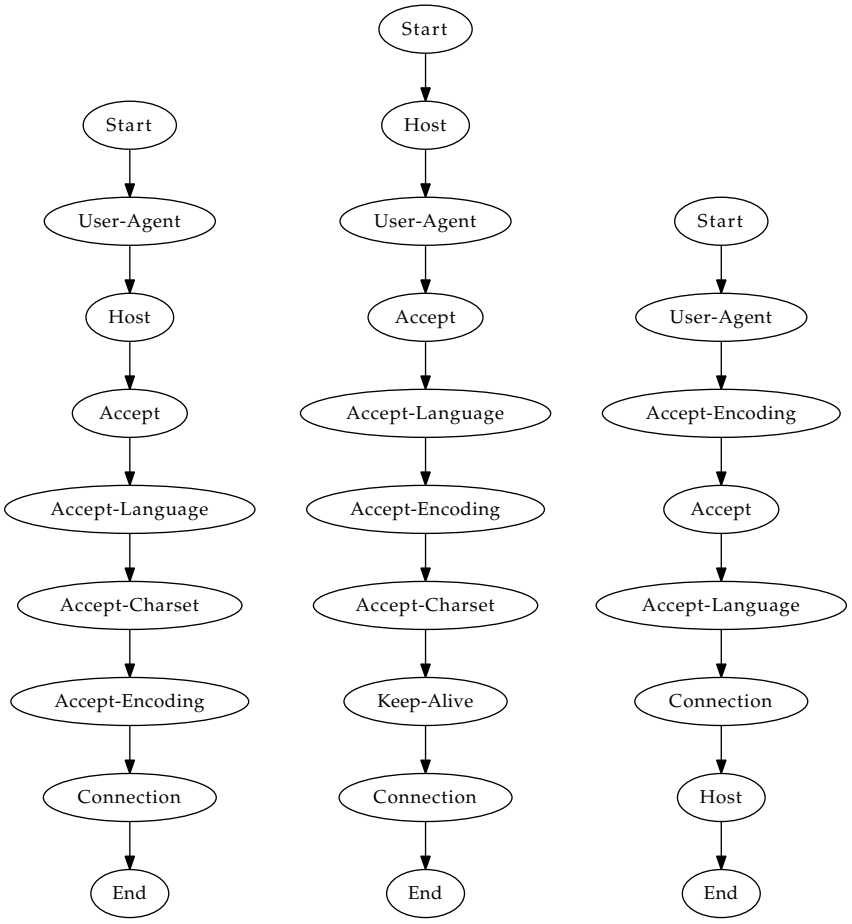



(a) IE 5 (b) IE 5 XMLHTTPRequest (c) IE 6




(d) IE 7 (e) IE 8 & 9 (f) Maxthon 2

Figure B.3: Internet Explorer & Maxthon fingerprints




(a) Opera 9,10 & 11 (b) Seamonkey 1 (c) Safari 3

(d) Safari 4 (e) Safari 5

Figure B.4: Opera, Safari & Seamonkey fingerprints

# BIBLIOGRAPHY


[1] Amada. URL http://amada.abuse.ch.

[2] Browserrecon project. URL http://www.computec.ch/projekte/browserrecon/.

[3] Contagio. URL http://contagiodump.blogspot.com/.

[4] URL http://www.damballa.com/solutions/damballa_firstalert.php.

[5] Internet simulator. URL www.inetsim.org.

[6] Malware domain list, . URL http://www.malwaredomainlist.com.

[7] Malware domains, . URL http://www.malwaredomains.com.

[8] Metasploit framework. URL www.metasploit.org.

[9] Offensive computing. URL www.offensivecomputing.net.

[10] Opendpi. URL www.opendpi.org.

[11] tcpdump. URL www.tcpdump.org.

[12] Zeus tracker. URL http://zeustracker.abuse.ch.

[13] Annual global threat report 2009. Technical report, Scansafe, 2010. URL http://www.scansafe.com/downloads/gtr/2009_AGTR.pdf.

[14] D. Anselmi and R. et al. Boscovich. Security intelligence report. Technical report, Microsoft, 2010.

[15] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for dns. In *19th Usenix Security Symposium*, 2010.

[16] D. Ariu, R. Tronci, and G. Giacinto. Hmmpayl: An intrusion detection system based on hidden markov models. *Computers & Security*, 2011.

[17] B. AsSadhan, J.M.F. Moura, D. Lapsley, C. Jones, and W.T. Strayer. Detecting botnets using command and control traffic. In *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pages 156–162. IEEE, 2009.

[18] J.M. Bauer, J.G. Michel, and Y. Wu. Itu study on the financial aspects of network security: Malware and spam. *ICT Applications and Cybersecurity Division, International Telecommunication Union, Final Report, July*, 2008.

[19] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. A view on current malware behaviors. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, pages 8–8. USENIX Association, 2009.

[20] J.R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, pages 7–7, 2006.

[21] H. Bos, M. van Steen, and N. Pohlmann. On botnets that use dns for command and control.

[22] A. Caglayan, M. Toothaker, D. Drapeau, D. Burke, and G. Eaton. Real-time detection of fast flux service networks. In *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 285–292. IEEE Computer Society, 2009.

[23] A. E. Cha and E. Nakashima. Google china cyberattack part of vast espionage campaign, experts say. The Washington Post, January 14 2010.

[24] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet detection by monitoring group activities in dns traffic. pages 715–720, October 2007. doi: 10.1109/CIT.2007.90. URL `http://dx.doi.org/10.1109/CIT.2007.90`.

[25] S. Coyne and R. Kazanciyan. The getaway. Black Hat DC 2011, January 2011.

[26] R. Deibert, A. Manchanda, R. Rohozinski, N. Villeneuve, and G. Walton. Tracking ghostnet: Investigating a cyber espionage network. *Information Warfare Monitor, Munk Centre, JR02-2009, March*, 29, 2009.

[27] I.S. Dhillon, S. Mallela, and R. Kumar. A divisive information theoretic feature clustering algorithm for text classification. *The Journal of Machine Learning Research*, 3:1265–1287, 2003.

[28] M. Dusi, A. Este, F. Gringoli, and L. Salgarelli. Using gmm and svm-based techniques for the classification of ssh-encrypted traffic. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[29] J. Finkle. Spain busts ring accused of infecting 13 mln pcs, March 2 2010. URL `http://www.reuters.com/article/2010/03/02/us-crime-hackers-idUSTRE6214ST20100302`.

[30] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the 15th USENIX Security Symposium*, pages 241–256, 2006.

[31] M. Fossi, G. Egan, K. Haley, E. Johnson, T. Mack, T. Adams, J. Blackbird, M.K. Low, D. Mazurek, D. McKinney, and P. Wood. Symantec internet security threat report. *XVI, April*, 2011.

[32] J. François, S. Wang, R. State, and T. Engel. Bottrack: Tracking botnets using netflow and pagerank. *NETWORKING 2011*, pages 1–14, 2011.

[33] T. Gentilcore. Going beyond gzipping. Velocity, 2009.

[34] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Recent Advances in Intrusion Detection*, pages 326–345. Springer, 2009.

[35] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 8–8. USENIX Association, 2007.

[36] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, page 12. USENIX Association, 2007.

[37] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium*, pages 139–154. USENIX Association, 2008.

[38] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium*. Citeseer, 2008.

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

[40] G. Hogben, D. Plohmann, E. Gerhards-Padilla, and F. Leder. Botnets: Detection, measurement, disinfection & defence. Technical report, European Network and Information Security Agency (ENISA), 2011.

[41] The Honeynet-Project. Know your enemy: Fast-flux service networks, an ever changing enemy. Technical report, The Honeynet Project & Research Alliance, 2007. URL `http://www.honeynet.org/papers/ff/`.

[42] C.H. Hsu, C.Y. Huang, and K.T. Chen. Fast-flux bot detection in real time. In *Recent Advances in Intrusion Detection*, pages 464–483. Springer, 2011.

[43] C.W. Hsu, C.C. Chang, C.J. Lin, et al. A practical guide to support vector classification, 2003.

[44] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks*, 51:1239–1255, April 2007. ISSN 1389-1286. doi: 10.1016/j.comnet.2006.09.016. URL `http://dl.acm.org/citation.cfm?id=1224244.1224379`.

[45] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 7–7. USENIX Association, 2007.

[46] R. Kazanciyan. The state of the hack. Tampa Bay ISSA InfraGard, December 2010.

[47] S. Kondo and N. Sato. Botnet traffic detection techniques by c&c session classification using svm. *Advances in Information and Computer Security*, pages 91–104, 2007.

[48] B. Krebs. N.y. firm faces bankruptcy from $164,000 e-banking loss, February 2010. URL `http://krebsonsecurity.com/2010/02/n-y-firm-faces-bankruptcy-from-164000-e-banking-loss/`. [Online; accessed 27-Juny-2011].

[49] C. Livadas, R. Walsh, D. Lapsley, and W.T. Strayer. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN Workshop on Network Security*. Citeseer, 2006.

[50] W. Lu, G. Rammidi, and A.A. Ghorbani. Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, 2010.

[51] C. McCarthy. An investigation on detecting applications hidden in ssl streams using machine learning techniques. 2010.

[52] S.K. Noh, J.H. Oh, J.S. Lee, B.N. Noh, and H.C. Jeong. Detecting p2p botnets using a multi-phased flow model. In *Proceedings of the 2009 Third International Conference on Digital Society*, pages 247–253. IEEE Computer Society, 2009.

[53] R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 488–498, Washington, DC, USA, 2006. IEEE Computer Society. doi: 10.1109/ICDM.2006.165. URL `http://dx.doi.org/10.1109/ICDM.2006.165`.

[54] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864–881, 2009.

[55] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 26–26. USENIX Association, 2010.

[56] M. Polychronakis, K.G. Anagnostakis, and E.P. Markatos. Emulation-based detection of non-self-contained polymorphic shellcode. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, pages 87–106. Springer-Verlag, 2007.

[57] I. Ristic. State of ssl. InfoSec World 2011, April 2011.

[58] S. Santorelli. Developing botnets. Technical report, Team Cymru, January 2010.

[59] K. Selvaraj and N. F. Gutierrez. The rise of pdf malware. Technical report, Symantec, 2010.

[60] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the storm and nugache trojans: P2p is here. *USENIX; login*, 32(6):2007–12, 2007.

[61] R. Villamarín-Salomón and J.C. Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 476–481. IEEE, 2008.

[62] N. Villeneuve. Command and control in the cloud, October 2010. URL `http://www.nartv.org/2010/10/22/command-and-control-in-the-cloud/`.

[63] K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[64] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.

[65] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. *Computer Security–ESORICS 2009*, pages 232–249, 2009.

[66] H. Xiong, P. Malhotra, D. Stefan, C. Wu, and D. Yao. User-assisted host-based detection of outbound malware traffic. In *Information and Communications Security: 11th International Conference, Icics 2009*, volume 5927, page 293. Springer-Verlag New York Inc, 2009.

[67] T.F. Yen and M.K. Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *2010 International Conference on Distributed Computing Systems*, pages 241–252. IEEE, 2010.

[68] T.F. Yen, X. Huang, F. Monrose, and M. Reiter. Browser fingerprinting from coarse traffic summaries: Techniques and implications. *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 157–175, 2009.