Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Pavlos Lamprakis

Human or malware?
Detection of malicious Web requests

Master Thesis MA-2016-48
March 2016 to September 2016

Tutor: Dr. David Gugelmann, Networked Systems Group
Co-Tutor: Dr. Markus Happe, Networked Systems Group
Supervisor: Prof. Dr. Laurent Vanbever, Networked Systems Group

Acknowledgments

At this point, I would like to thank my advisors Dr. David Gugelmann and Dr. Markus Happe for their valuable insights, guidance and for the pleasant and motivating atmosphere they created during our meetings throughout the duration of my thesis. Without their help this thesis would not have been possible.

Furthermore, I am grateful to my supervisor, Prof. Dr. Laurent Vanbever, for providing me with the opportunity to conduct this master thesis in the Networked Systems Group and for his critical remarks after my intermediate presentation.

Zurich, September 2016

Pavlos Lamprakis

Abstract

Nowadays covert command and control (C&C) communication channels are built using the HTTP/HTTPS protocol, mainly because it is rarely blocked as well as malicious traffic can hide inside huge amounts of daily benign browsing traffic. This thesis addresses the problem of identifying malicious Web traffic and more specifically, post-infection traffic (C&C communication). We have built a system to facilitate network traces' analysis by combining different existing tools. We collected and classified a large number of benign and malicious network traces. Using this system, we performed an extensive analysis of these traces and found common patterns occurring in them. Based on our analysis, we found that C&C communication can be reliably detected by representing the dependencies of HTTP/HTTPS traffic in a graph and complementing missing links. As a result, C&C traffic stands out as unconnected nodes. We applied different classifiers on the graph and found that a Gradient Boosting classifier can detect C&C traffic with 99% precision and 97% recall.

Contents

1 Introduction **11**
1.1 Motivation . . . 11
1.1.1 Cyberespionage against RUAG . . . 11
1.1.2 Cyber attack on RSA Security . . . 12
1.2 The task . . . 13
1.3 Related work . . . 13
1.4 The lifecycle of an infection . . . 14
1.5 Overview . . . 14

2 System Architecture **17**
2.1 Existing tools . . . 17
2.1.1 Browsing traffic generator . . . 17
2.1.2 Capturing network packets . . . 17
2.1.3 Tools to intercept SSL/TLS traffic . . . 18
2.1.4 Bro IDS . . . 18
2.1.5 Hviz . . . 20
2.2 Our traffic analysis system architecture . . . 23

3 Baseline Analysis **25**
3.1 Datasets . . . 25
3.1.1 Benign traffic . . . 25
3.1.2 Malicious traffic . . . 26
3.2 Some special malware C&C cases . . . 26
3.3 The Content-Type header field . . . 27
3.4 Unrelated nodes . . . 29
3.4.1 Unrelated nodes in malicious traffic . . . 29
3.4.2 Unrelated nodes in benign traffic . . . 29
3.4.3 Why unrelated nodes exist in benign traffic . . . 30

4 Building a Request Classifier **33**
4.1 Adding the missing links in the graph . . . 33
4.1.1 The algorithm in high-level . . . 33
4.2 Simple classifier using only unrelated nodes . . . 37
4.2.1 Evasion techniques . . . 37
4.3 Robust classifier . . . 39
4.3.1 Feature selection . . . 39

5 Evaluation **41**
5.1 Evaluation of the graph completion algorithm . . . 41
5.1.1 Measuring the ability to reduce the number of unrelated nodes . . . 41
5.1.2 Injection of unrelated nodes of benign traffic into benign traffic . . . 41

5.1.3 Injection of unrelated nodes of malicious traffic into benign traffic . 43
5.1.4 Discussion 43
5.2 Evaluation of the classifiers 44
5.2.1 Evaluation Setup 44
5.2.2 Evaluation of the classifier using only unrelated nodes 45
5.2.3 Evaluation of the robust classifiers 45

6 Conclusion **47**
6.1 Summary 47
6.2 Outlook 47

List of Figures

1.1 Exploit-Kit Attack Scenario - Adapted from [2]. 15

2.1 Parts of the dns.log of a host infected by Zeus trojan horse malware. 19
2.2 Parts of the http.log of a host infected by Zeus trojan horse malware. . . . 20
2.3 The request graph by Hviz from a Zeus malware variant. 22
2.4 Examination of suspicious requests from a Zeus malware variant. 22
2.5 The process of analyzing a libpcap file. 23

3.1 The top 10 content types found in benign traffic. 28
3.2 The top 10 content types found in malicious traffic. 28

4.1 The time-window of an unrelated node's possible parents. 34
4.2 The top-5 Content-Types loaded from an HTML page. 35
4.3 The top-5 Content-Types loaded from a CSS file. 36
4.4 The estimated pdf of unrelated nodes in benign and malicious samples of the training set, after the execution of the graph completion algorithm. 38

5.1 Evaluation setup - Injection of unrelated nodes of benign traffic into benign traffic. 42

List of Tables

3.1 Suspicious domain names from a variant of Sality malware. 29

5.1 The reduction of unrelated nodes by executing the algorithm on 5 random network traces. 41
5.2 Injection of unrelated nodes of benign traffic into benign traffic. 42
5.3 Injection of unrelated nodes of malicious traffic into benign traffic. 43
5.4 Evaluation of classifier using only unrelated nodes over the test set. 45
5.5 Optimal feature selection which maximizes the f1-score. 45
5.6 Maximum average f1-score obtained by 5-fold cross-validation over the training set for both classes. 45
5.7 Evaluation of different classifiers over the test set. 46

Chapter 1

Introduction

Web traffic has become the ideal choice of communication with infected hosts for various reasons. First and foremost, even security-aware corporate networks have not that strict policies for inbound and outbound HTTP/HTTPS traffic as they need to allow their employees to browse the Web. Moreover, HTTP/HTTPS channels have clear advantages over some popular alternative options. They outperform Peer to Peer (P2P) channels, as they are simpler and easier to use as well as IRC channels, since they are more resilient. The latter is achieved by using IP and domain fluxing which increases the possibility of compromised hosts to locate an online C&C server. In addition, P2P and IRC protocols are blocked by companies and their traffic is suspicious. Finally, an HTTP/HTTPS channel can be considered covert, because during everyday web browsing a huge number of requests and responses are being exchanged. For instance, one user click to load http://www.20min.ch, the popular Swiss news website, triggers approximately 400 HTTP/HTTPS requests. Consequently, malicious traffic can hide inside benign traffic and, without appropriate measures, avoid detection.

Cyber criminals use HTTP/HTTPS command and control (C&C) servers, in order to communicate with compromised hosts, also known as zombies. Those hosts are primarily used for exfiltrating sensitive data (e.g., personal information, credit card numbers, industrial secrets), relaying traffic, gaining access to neighboring hosts inside their internal network (pivoting), etc.

1.1 Motivation

To give an idea of the importance of the problem as well as the disastrous effects it can cause, we present two relevant case studies we found interesting during our research.

1.1.1 Cyberespionage against RUAG

RUAG [30] is a technology company owned by the Swiss government that develops and supplies military equipment. This firm was compromised and its case was made public very recently (May 2016). According to security analysts from the Reporting and Analysis Center for Information Assurance MELANI, who published a report [27] about that case: "the cyber attack was related to a long running campaign of the threat actor around Epic/Turla/Tavdig. The actor has not only infiltrated many governmental organizations in Europe, but also commercial companies in the private sector in the past decade". The attackers followed a usual pattern where they first infected some easy to target hosts and afterwards, they moved laterally in the network by infecting additional devices and elevating their privileges, in order to acquire the sensitive information they wanted to exfiltrate.

The chronology of the attack adapted from [27] is the following:

- **9.2014**: Indicators of compromise (IOCs) already appear in proxy logs. However, they were not detected at that time. Since there are not any proxy logs available before this date, the initial vector is still unknown.
- **12.2015**: Some suspicious IPs from an external organization appear in the logs. However, an in-depth search was not possible at that point because the proxy did not log internal client IPs.
- **21.1.2016**: A major incident was opened by MELANI/GovCERT.ch and RUAG because C&C servers and infected bots were identified in the proxy logs.
- **22.1-31.1.2016**: This is the hot phase of the incident response and a task-force has been established. The investigators performed forensic analysis of logs, disks etc. At that point it was identified that the attack goes back to September 2014 where logs end as well as that Turla/Tavdig Malware has been used.
- **1.2-29.2.2016**: Monitoring has been established and several new C&C servers were found. Moreover, exfiltration using HTTP over proxy in waves was found to have taken place in June, July, September, October and December in 2015.
- **1.3-30.4.2016**: Enhanced monitoring was established.
- **3.5.2016**: Several press reports about the incident were made public. This leakage heavily damaged the ongoing investigation rendering the ongoing monitoring useless.

The reason why this case is relevant is because the malware exfiltrated stolen data to the outside world using HTTP POST requests. The total size of those data was approximately 23GB, however, this number cannot be estimated with high accuracy since they were firstly compressed and then uploaded, some files were exfiltrated more than once and that number includes beaconing requests to the C&C servers, as well.

1.1.2 Cyber attack on RSA Security

RSA Security LLC [29] is a USA-based company dealing with computer and network security. It provides its worldwide customers with the essential security solutions to protect themselves from cyber atttacks. One of its main products is the SecurID authentication token which is used for two-factor authentication.

In 2011 the enterprise reported that they suffered a data breach and the anatomy of the attack was similar to the RUAG Case described above. The attackers firstly targeted specific employees using social engineering attacks. In particular, they sent spear phishing emails with a malicious Excel file attached to them. When an RSA employee opened that file, the malware exploited a zero-day vulnerability in Adobe Flash. After gaining access to one of RSA's machines, the attackers installed a difficult to spot Poison Ivy RAT variant, in order to remotely control the internal machine. Subsequently, they pivoted inside the network and gained access to more valuable machines. Finally, they exfiltrated the gathered sensitive data to a remote compromised machine [28].

This breach cost the parent company, EMC, $66 million [14] because it had to replace the authentication tokens of its customers. The reason was that among the data leaked, there was also a database containing the mapping of token serial numbers to secret token seeds which would allow one to predict the generated sequence of tokens and to bypass the second factor.

1.2 The task

Being able to identify malicious traffic inside benign traffic can be a really complex task. In order to support investigators, tools like Hviz [18] have been developed. Hviz' main focus is on visualizing the timeline of a client's Web browsing, that is, to reconstruct the sequence of Web pages visited by a user from network traces. It allows an investigator to quickly understand the context of a security alert. This can considerably speed up the bootstrapping of a forensic investigation because an analyst often starts an investigation with only very limited information (such as an IDS alert pointing to a supposedly infected client). A detailed description of Hviz can be found in section 2.1.5.

However, Hviz does not detect malicious network activities. In this thesis, we investigate methods for detecting C&C traffic in HTTP/HTTPS. In a few words, this goal is achieved by dividing the final task into the following sub-tasks. Firstly, we performed background research on existing Web-based malware. Furthermore, we analyzed typical sequences of events occurring during human Web browsing and compared them with the ones caused by a malware running on a compromised host. Last but not least, we developed and evaluated a detector for C&C traffic. Those functions can be added as an additional layer to a network analysis system like Hviz and improve it. With all the above completed, the system could assist a forensic investigator and be capable of classifying whether a Web request was due to a human or a malware.

1.3 Related work

Burghouwt et al. (2011) [16] address the detection of social media C&C traffic. They performed their evaluation with a malware that uses Twitter as C&C channel. The main concept of this research is that whenever traffic from Twitter is observed, a validation step is performed. It is checked whether within a small time-window there was also an event from the mouse or the keyboard (e.g., press of F5 key to reload the page, or enter) which could justify that traffic. However, their approach relies on recording user activity such as mouse clicks and keyboard strokes together with network traffic.

ExecScent [25] can mine new previously unknown C&C domain names from network traffic. The developers trained their classifier by grouping together similar C&C requests in the following way. They detected strings that represent data of a certain type and replaced them with tags using their data type and string length. For better understanding, an example transformation is shown below:

```
GET /Ym90bmV0DQo=/cnc.php?v=121&cc=IT
GET /<Base64;12>/cnc.php?v=<Int;3>&cc=<Str;2>
```

Using that method, they managed to create "high quality" clusters and thus, to identify suspicious requests by comparing the corresponding distances to them.

In their system called WebWitness [26], Nelms et al. (2015) reconstruct a client's Web activity by building a request graph similar to Hviz (see 2.1.5). In their situation, however, the graph is weighted and the weights are added depending on the possibility that two nodes are connected. Their goal is to find the cause (e.g., drive-by, social engineering or update/drop for all the remaining cases) of a malicious executable file download. When there is a node which corresponds to a download, the graph is traversed backwards, following the highest weighted edges which lead to the actual cause. We use a quite similar approach for complementing the request graph, however in contrast to [26], we show that this approach cannot only identify the cause of known infections, but also detect new, unknown infections based on the C&C traffic.

Kim et al. (2014) have designed and implemented HAS-Analyzer [23], a system that detects HTTP-based C&C servers using network traffic only. For each client, they extract the trees that contain the head requests (user clicks) and the subsequent embedded ones such as requests to load third-party content. Their analysis showed that an HTTP-based C&C uses a small portion of network resources in order to stay undetected and that the content of a request or a response is changed frequently among consecutive C&Cs. Based on those two observations, they built a classifier that has approximately 96% accuracy, 1.3% false positives and 5% false negatives without using any whitelists.

1.4 The lifecycle of an infection

There are several ways a host can get infected with a malware. In this section, we describe a popular method (depicted in figure 1.1). A user visits a compromised website which has a (hidden) malicious iFrame injected or a legitimate one where a malicious advertisement is embedded. The iFrame/advertisement performs a drive-by attack by redirecting the victim to a malicious server which hosts an exploit kit. The latter will scan the victim's system for vulnerabilities and try to exploit them. If the exploit is successful, the malicious payload is run and then the victim becomes infected with a malware which in turn can download various kinds of malicious tools. Finally, the post-infection traffic follows where the malware communicates with the C&C server to receive commands to execute.

1.5 Overview

Chapter 2 introduces various tools that we have tested and describes how they were combined, in order to build a traffic analysis system. In chapter 3 we present how we obtained the datasets used, we compare benign with malicious traffic and explain common patterns in them. In chapter 4 we introduce an algorithm which we developed to improve the quality of the request graph as well as the process of building a robust classifier that is able to distinguish between benign and malicious traffic. In chapter 5, the developed algorithm and classifiers are evaluated with different setups. Finally, in chapter 6, we summarize our work, state interesting open problems and offer some suggestions for future work.

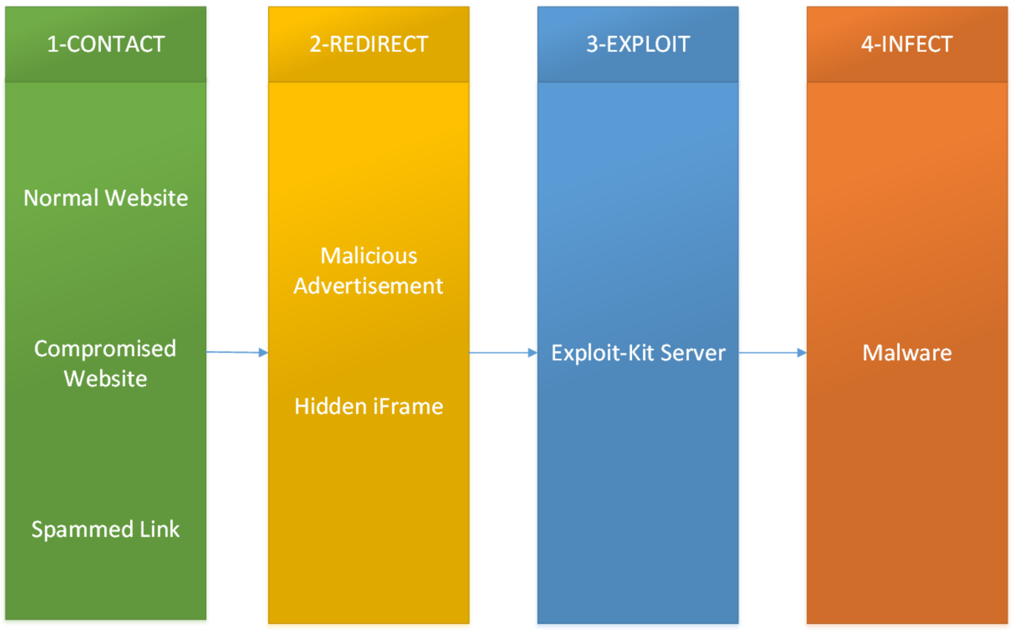


Figure 1.1: Exploit-Kit Attack Scenario - Adapted from [2].

Chapter 2

System Architecture

In this chapter, we describe how we combined and used existing tools, in order to build a system for detecting C&C traffic over HTTP/HTTPS. Firstly, we are introducing each one of them and subsequently, we present the system architecture we have built.

2.1 Existing tools

The following tools belong to the classes of simulating human Web browsing, recording network traffic, intercepting SSL/TLS and analyzing network traces. All of them were tested in order to pick the appropriate ones for our system.

2.1.1 Browsing traffic generator

In order to efficiently collect a sufficient amount of traffic to perform our analysis, a Python script developed for Hviz [18] which simulates human Web browsing was used. This script is based on Selenium-WebDriver [31] which has an API that makes it easier to automate Web browsing. This is the case because the Selenium-WebDriver API "performs direct calls to the browser using each browser's native support for automation". Even though this script uses Firefox, this is not binding. It would be possible with minor adjustments to record traffic using other browsers, since the WebDriver is compatible with most of them.

The script takes as input a list of websites which visits in random order. On each one of them it makes 5 clicks per average and stays to each resulting page for a random time interval. The time spent on each page has an upper bound of 30 seconds. What is more, it provides the option to record only unencrypted HTTP traffic. This is achieved by visiting the HTTP versions of the websites included in the input list. In case the website is forcing SSL by redirecting the client to its secure version, the connection is terminated and the next URL in the list is fetched. Otherwise, the script continues imitating a human.

2.1.2 Capturing network packets

Tcpdump

Tcpdump [33] is a popular command-line packet analyzer which uses the libpcap library to capture packets. While the simulator is executing, tcpdump needs to be running in the background at the same time, in order to record the generated traffic. Tcpdump is easy to use and its output (in libpcap format) can be parsed and analyzed by Wireshark [13] and BroIDS [10]. The latter is described in section 2.1.4.

2.1.3 Tools to intercept SSL/TLS traffic

SSLStrip2 & dns2proxy

SSLStrip [32] was introduced in 2009, when its developer, Moxie Marlinspike, presented it at Black Hat DC. Using this tool during a man-in-the-middle (MITM) attack, a network attacker could ”prevent a web browser from upgrading to an SSL connection in a subtle way that would likely go unnoticed by a user” [32]. The HTTP Strict Transport Security (HSTS) specification [19] was subsequently developed to counter these attacks. In order to bypass HSTS, *SSLStrip2* (an updated version of SSLStrip written by another developer) in combination with *dns2proxy* [7] (a DNS server proxy) has to be used.

SSLSplit

SSLSplit [9] is a tool that performs MITM attacks against SSL/TLS encrypted network connections. It intercepts connections transparently through a network address translation engine and redirects them to itself. ”SSLsplit terminates SSL/TLS and initiates a new SSL/TLS connection to the original destination address, while logging all data transmitted” [9]. This is accomplished by generating a certificate on-the-fly and signing it with the private key of a CA certificate that the client has to already trust. It supports plain TCP, plain SSL, HTTP and HTTPS connections over both IPv4 and IPv6.

Mitmdump-Mitmproxy

As their developers describe briefly on their website [5], ”mitmproxy is an interactive console program that allows traffic flows to be intercepted, inspected, modified and replayed” whereas mitmdump is actually ”tcpdump for HTTP”. Those tools perform the same kind of man-in-the-middle attack on SSL/TLS as SSLSplit, however they support more features (e.g., traffic flows can be intercepted, inspected, modified and replayed).

The three different intercepting options described above were evaluated and the one that suited best our needs was selected. Mitmdump was used to record SSL/TLS traffic, whereas mitmproxy to inspect and understand it. Apart from the fact that they are both very easy to install and run, their decisive advantage the others lack is that they have a powerful scripting API. This API is ”event driven - a script is simply a Python module that exposes a set of event methods” [5]. It provides us with objects which hold each HTTP request and corresponding HTTP response. Since Hviz already supports parsing mitmdump files using the above API, mitmdump was the ideal choice for our situation. The reason why the other options were not chosen are the following. SSLStrip & dns2proxy would not be useful since the same result could be achieved much easier by configuring our script to visit only HTTP webpages. In addition, websites that do not allow HTTP connections would not be visited which means that our requirement to record SSL traffic is not fulfilled. Last but not least, this method is never used in enterprise networks where usually a web intercepting proxy that allows for SSL MITM is used. However, it is still an interesting approach as it could be and is used for network penetration. As for SSLSplit, even though it is simple and efficient (written in C), the main disadvantage of it is that its logs require a lot of effort to be parsed in comparison to the ones of mitmdump.

2.1.4 Bro IDS

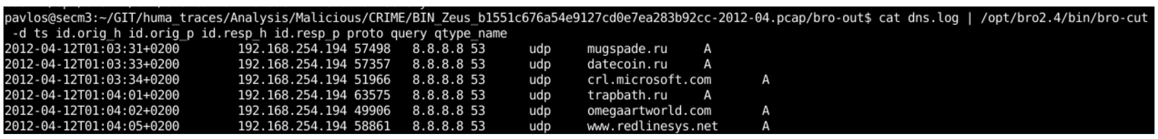
Bro IDS [10] is a powerful network analysis framework which provides many functionalities. Bro’s output is a large amount of useful logs extracted by parsing the input network trace. Some of them are output by default but one could write scripts using the Bro language in order to create logs of their choice. Some of the most useful logs that Bro outputs are the following:

- **conn.log**: contains the IP, TCP, UDP and ICMP connection details such as the originating and responding endpoint's IP address and port, the protocol used, the size of the sent and received payload, etc.
- **dns.log**: contains the DNS activity such as the domain name of the query, the query type (e.g., A, AAAA, PTR), the kind of answer (e.g., authoritative), etc.
- **ftp.log**: contains the FTP session-level activity such as the username and the password for the FTP session, the commands issued by the client, the size of the transferred file, etc.
- **files.log**: contains information about files transferred over the network such as the file name and type, its size, its md5/sha1/sha256 hash, etc. This information is aggregated from different protocols, including HTTP, FTP, and SMTP.
- **http.log**: contains information about the http requests and corresponding responses such as the URI and the method (e.g., GET, POST, HEAD, etc.) used in the request, the value of the referer header, the status code returned by the server, the value of the User-Agent header, etc.
- **ssl.log**: contains information about the SSL handshakes such as the SSL version that the server offered, the SSL cipher used, the MD5 hash of the raw server certificate, etc.
- **software.log**: contains information about the software that are being used on the network such as the IP address of the host running it, the port which it is listening, its name, its type, its version, etc.

All of the above logs are useful for the analysis of a network trace since they separate the information it contains in a meaningful way. Moreover, those logs can easily be parsed by pandas [8], the well known data analysis library for python, in order to extract useful results from them. However, in order to create the request graph (see 2.1.5), only the *http.log* in addition to some extra information for each HTTP request/response extracted by Bro scripts had to be used.

Bro IDS - Zeus case study

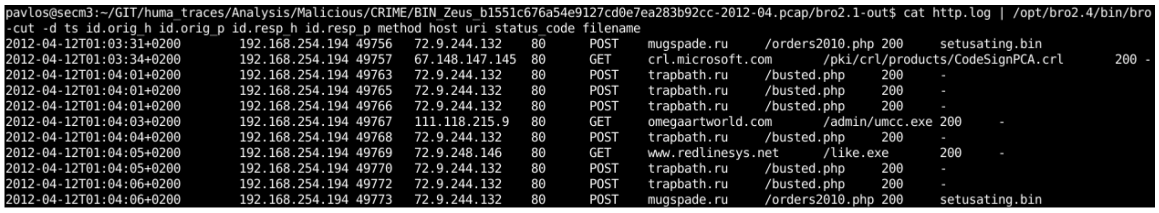
In order for the reader understand better and get an intuition of the output logs of Bro IDS as well as their importance, this section shows an example usage of it. In figure 2.1 some fields of the *dns.log* of Bro are shown. A host with the IP address 192.168.254.194 contacted Google's public DNS server asking for the A record of among others, some suspicious Russian domain names.



```
pavlos@secm3:~/GIT/huma_traces/Analysis/Malicious/CRIME/BIN_Zeus_b1551c676a54e9127cd0e7ea283b92cc-2012-04.pcap/bro-out$ cat dns.log | /opt/bro2.4/bin/bro-cut
 -d ts id.orig_h id.orig_p id.resp_h id.resp_p proto query qtype_name
2012-04-12T01:03:31+0200        192.168.254.194 57498   8.8.8.8 53      udp     mugspade.ru     A
2012-04-12T01:03:33+0200        192.168.254.194 57357   8.8.8.8 53      udp     datecoin.ru     A
2012-04-12T01:03:34+0200        192.168.254.194 51966   8.8.8.8 53      udp     crl.microsoft.com       A
2012-04-12T01:04:01+0200        192.168.254.194 63575   8.8.8.8 53      udp     trapbath.ru     A
2012-04-12T01:04:02+0200        192.168.254.194 49906   8.8.8.8 53      udp     omegaartworld.com       A
2012-04-12T01:04:05+0200        192.168.254.194 58861   8.8.8.8 53      udp     www.redlinesys.net      A
```

Figure 2.1: Parts of the dns.log of a host infected by Zeus trojan horse malware.

To investigate the issue further, an investigator could examine the HTTP activity inside the *http.log* file. This is depicted in figure 2.2.



```
pavlos@secm3:~/GIT/huma_traces/Analysis/Malicious/CRIME/BIN_Zeus_b1551c676a54e9127cd0e7ea283b92cc-2012-04.pcap/bro2.1-out$ cat http.log | /opt/bro2.4/bin/bro
-cut -d ts id.orig_h id.orig_p id.resp_h id.resp_p method host uri status_code filename
2012-04-12T01:03:31+0200        192.168.254.194 49756   72.9.244.132    80      POST    mugspade.ru     /orders2010.php 200     setusating.bin
2012-04-12T01:03:34+0200        192.168.254.194 49757   67.148.147.145  80      GET     crl.microsoft.com       /pki/crl/products/CodeSignPCA.crl       200 -
2012-04-12T01:04:01+0200        192.168.254.194 49763   72.9.244.132    80      POST    trapbath.ru     /busted.php     200     -
2012-04-12T01:04:01+0200        192.168.254.194 49765   72.9.244.132    80      POST    trapbath.ru     /busted.php     200     -
2012-04-12T01:04:01+0200        192.168.254.194 49766   72.9.244.132    80      POST    trapbath.ru     /busted.php     200     -
2012-04-12T01:04:03+0200        192.168.254.194 49767   111.118.215.9   80      GET     omegaartworld.com       /admin/umcc.exe 200     -
2012-04-12T01:04:04+0200        192.168.254.194 49768   72.9.244.132    80      POST    trapbath.ru     /busted.php     200     -
2012-04-12T01:04:05+0200        192.168.254.194 49769   72.9.248.146    80      GET     www.redlinesys.net      /like.exe       200     -
2012-04-12T01:04:05+0200        192.168.254.194 49770   72.9.244.132    80      POST    trapbath.ru     /busted.php     200     -
2012-04-12T01:04:06+0200        192.168.254.194 49772   72.9.244.132    80      POST    trapbath.ru     /busted.php     200     -
2012-04-12T01:04:06+0200        192.168.254.194 49773   72.9.244.132    80      POST    mugspade.ru     /orders2010.php 200     setusating.bin
```

Figure 2.2: Parts of the http.log of a host infected by Zeus trojan horse malware.

In this log the full URLs visited are shown. The host has performed a POST request and received as response a file with the filename *setusating.bin*. Actually, this POST request had the following POST parameters:

`btn1=WIN7PRO_X86_000_74DEB1E36522DF69_26&sk1=[HASH]`

This communication pattern is well-known for Zeus malware. The host performs the above POST request to the command and control server with those variables that represent the unique identifier of that host. Afterwards, the C&C server replies back with an encrypted configuration file with the above filename. Subsequently, after approximately 30 seconds, there are some POST requests to trapbath.ru/busted.php that are performed almost simultaneously and then the same communication with the C&C server described above.

What can also be discovered from that log file is that two executables with filenames *umcc.exe* and *like.exe* are being downloaded. Their md5 checksums can be found inside *files.log*. Both of them had more than 85 percent detection rate in VirusTotal [12], the popular web service that analyzes suspicious files and URLs.

From all the above findings we can conclude that this traffic is indeed malicious and that Bro IDS' logs are very useful for this kind of investigations.

2.1.5 Hviz

Hviz's [18] primary goal is to facilitate the tedious work during a forensic investigation of computer security incidents (e.g., online fraud, cyber crime, or data leakage). It achieves this by reconstructing and visualizing the Web browsing activity of individual hosts, in other words which websites a user visited, in a graph called the request graph. This graph is built using the NetworkX Python library [6] which is used "to create, manipulate and study the structure, dynamics and functions of complex networks". In this graph each node "represents an HTTP request and the corresponding HTTP response. If an HTTP request has a valid Referer header, there is a directed edge from the node corresponding to the Referer header to the node corresponding to the HTTP request" [18].

For instance, when a user visits `https://www.yahoo.com`, many follow-up requests and responses occur: the website will contact the appropriate CDN (e.g., Akamai, Amazon CloudFront) to load its static content (e.g., images, javascript files), advertisement and analytics providers, etc. Each request and response will be a node in the graph with attributes such as the timestamp of the request, its URL, the request and response headers, the length of the request/response etc. Those kind of requests that are issued without user involvement are called embedded requests. Each node that was a follow-up request/response caused by the initial one to `https://www.yahoo.com`, will be a destination node and a directed edge will be added between the causing and the resulting node. Furthermore, those embedded requests might trigger other (e.g., a CSS file might load other stylesheets as well as images) which will result in a more complex graph. The same would happen if after the page has loaded completely, the user clicks a hyperlink to `https://www.yahoo.com/news`. The request that is a consequence of a user click is called a head request.

Each node in the graph that Hviz builds is represented as a map data structure containing key-value pairs. Here are the most important attributes of a node that were mostly used during this thesis.

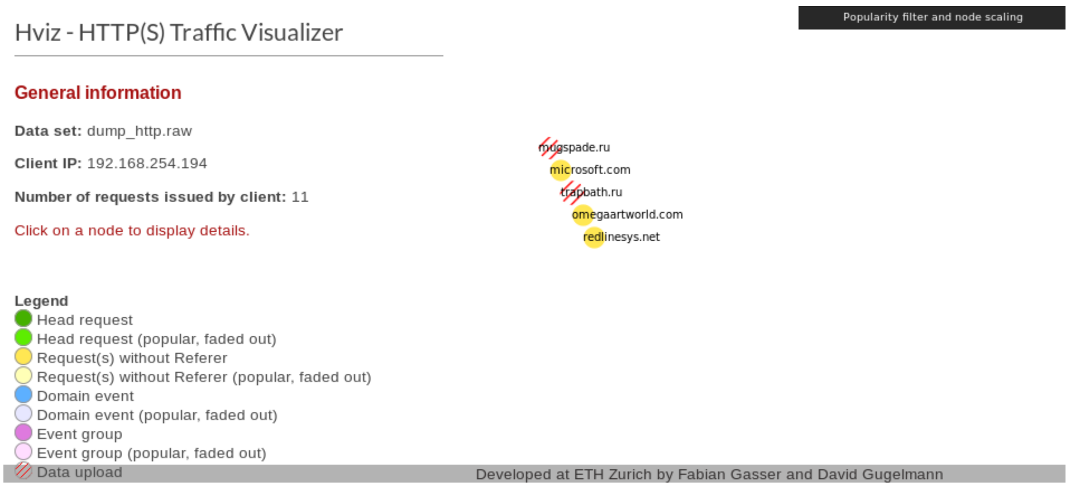
- **code**: the HTTP response status code (e.g., 200, 302, etc.)
- **req_URI**: the requested URL (e.g., https://www.google.com)
- **ts**: the timestamp that the request was performed
- **req_length**: the size of the request in bytes
- **res_length**: the size of the response in bytes
- **req_headers**: the HTTP headers of the request
- **res_headers**: the HTTP headers of the response
- **method**: the HTTP method used (e.g., GET, POST, HEAD, OPTIONS, etc.)
- **res_body_entities**: the body of the HTTP response (e.g., the HTML, Javascript, CSS code etc.), this field is only extracted for specific content types in order not to make the graph's size huge
- **invalid_ref**: this field is True when the referer header field does not correspond to a request URI of any previous node and False otherwise
- **content_type**: the content type of the HTTP response extracted from the response headers
- **parent_URI**: the request URL of the parent node which is actually the value of the referer HTTP header

Furthermore, NetworkX provides us with methods which could offer additional information about the whole graph such as the number of nodes it is comprised which actually corresponds to the number of HTTP requests in a network trace. What is more, it can give us more details for each node such as the number of its neighbors which corresponds to the number of requests that have this node's request URI as a referer.

In order to understand how Hviz could assist forensic investigators with their analysis, some of its key characteristics need to be stated. Firstly, Hviz can distinguish with approximately 80% accuracy between head and embedded requests. It is actually marking each kind of request in the graph using different colors, in order to be visually easier to discern. Another feature worth mentioning is that it tries to reduce the number of displayed nodes in the graph by aggregating similar events, when this is possible. In a few words, Hviz groups together requests with the same effective second level domain (domain aggregation). In addition, it groups together third party domains that are the same for different pages of the same website. This results in an overall event reduction factor of 19. As a consequence, the graph that is displayed becomes simpler and thus, easier to analyze. Last but not least, Hviz tags popular and special events. For instance, if the same requests happen between multiple hosts in the same network they are probably harmless and marked accordingly. In addition to that, file uploads are considered to be special events, since they can conceal sensitive data exfiltration and thus, nodes representing them are tagged, as well.

Hviz - Zeus case study

The same case that was described in section 2.1.4 can be analyzed by Hviz, as well. The output request graph of Hviz is shown in figure 2.3. From there an investigator can see that there is some data exfiltration going on that Hviz has marked. To investigate the issue further, one can examine the suspicious requests with their attributes as it is depicted in figure 2.4. By viewing the request body attributes of them, it becomes obvious that the traffic is malicious as it was described in the previous section.


Popularity filter and node scaling
Hviz - HTTP(S) Traffic Visualizer
mugspade.ru
microsoft.com
trapbath.ru
omegaartworld.com
redlinesys.net

General information

Data set: dump_http.raw

Client IP: 192.168.254.194

Number of requests issued by client: 11

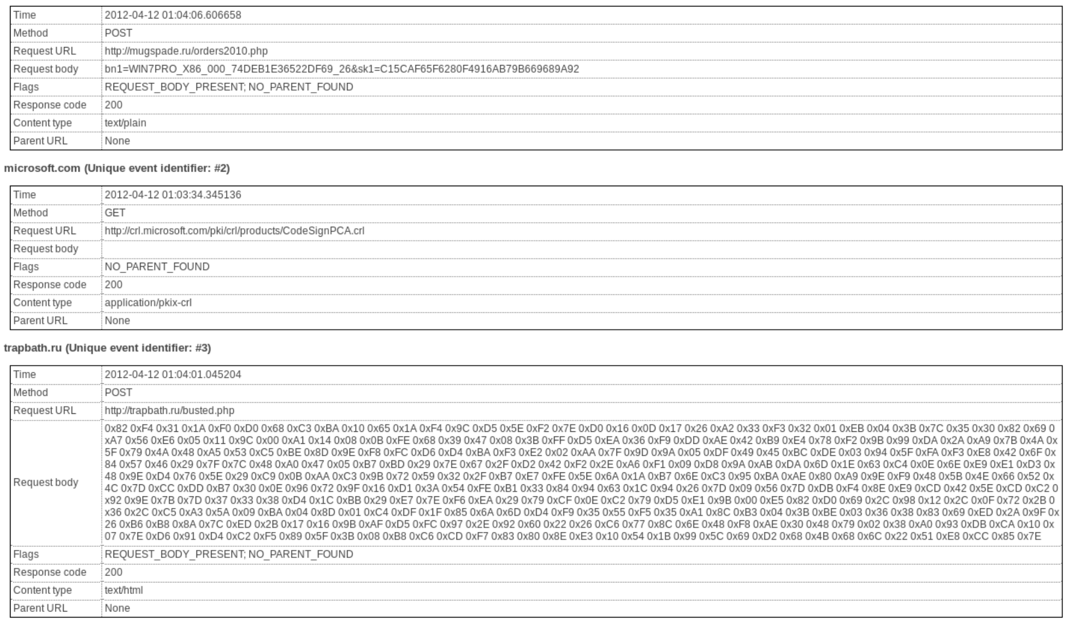
Click on a node to display details.

Legend

●	Head request
●	Head request (popular, faded out)
●	Request(s) without Referer
○	Request(s) without Referer (popular, faded out)
●	Domain event
○	Domain event (popular, faded out)
●	Event group
○	Event group (popular, faded out)
▨	Data upload

Developed at ETH Zurich by Fabian Gasser and David Gugelmann

Figure 2.3: The request graph by Hviz from a Zeus malware variant.



Time	2012-04-12 01:04:06.606658
Method	POST
Request URL	http://mugspade.ru/orders2010.php
Request body	bn1=WIN7PRO_X86_000_74DEB1E36522DF69_26&sk1=C15CAF65F6280F4916AB79B669689A92
Flags	REQUEST_BODY_PRESENT; NO_PARENT_FOUND
Response code	200
Content type	text/plain
Parent URL	None

microsoft.com (Unique event identifier: #2)

Time	2012-04-12 01:03:34.345136
Method	GET
Request URL	http://crl.microsoft.com/pki/crl/products/CodeSignPCA.crl
Request body	
Flags	NO_PARENT_FOUND
Response code	200
Content type	application/pkix-crl
Parent URL	None

trapbath.ru (Unique event identifier: #3)

Time	2012-04-12 01:04:01.045204
Method	POST
Request URL	http://trapbath.ru/busted.php
Request body	0x82 0xF4 0x31 0x1A 0xF0 0xD0 0x68 0xC3 0xBA 0x10 0x65 0x1A 0xF4 0x9C 0xD5 0x5E 0xF2 0x7E 0xD0 0x16 0x0D 0x17 0x26 0xA2 0x33 0xF3 0x32 0x01 0xEB 0x04 0x3B 0x7C 0x35 0x30 0x82 0x69 0xA7 0x56 0xE6 0x05 0x11 0x9C 0x00 0xA1 0x14 0x08 0x0B 0xFE 0x68 0x39 0x47 0x08 0x3B 0xFF 0xD5 0xEA 0x36 0xF9 0xDD 0xAE 0x42 0xB9 0xE4 0x78 0xF2 0x9B 0x99 0xDA 0x2A 0xA9 0x7B 0x4A 0x5F 0x79 0x4A 0x48 0xA5 0x53 0xC5 0xBE 0x8D 0x9E 0xF8 0xFC 0xD6 0xD4 0xBA 0xF3 0xE2 0x02 0xAA 0x7F 0x9D 0x9A 0x05 0xDF 0x49 0x45 0xBC 0xDE 0x03 0x94 0x5F 0xFA 0xF3 0xE8 0x42 0x6F 0x84 0x57 0x46 0x29 0x7F 0x7C 0x48 0xA0 0x47 0x05 0xB7 0xBD 0x29 0x7E 0x67 0x2F 0xD2 0x42 0xF2 0x2E 0xA6 0xF1 0x09 0xD8 0x9A 0xAB 0xDA 0x6D 0x1E 0x63 0xC4 0x0E 0x6E 0xE9 0xE1 0xD3 0x48 0x9E 0xD4 0x76 0x5E 0x29 0xC9 0x0B 0xAA 0xC3 0x9B 0x72 0x59 0x32 0x2F 0xB7 0xE7 0xFE 0x5E 0x6A 0x1A 0xB7 0x6E 0xC3 0x95 0xBA 0xAE 0x80 0xA9 0x9E 0xF9 0x48 0x5B 0x4E 0x66 0x52 0x4C 0x7D 0xCC 0xDD 0xB7 0x30 0x0E 0x96 0x72 0x9F 0x16 0xD1 0x3A 0x54 0xFE 0xB1 0x33 0x84 0x94 0x63 0x1C 0x94 0x26 0x7D 0x09 0x56 0x7D 0xDB 0xF4 0x8E 0xE9 0xCD 0x42 0x5E 0xCD 0xC2 0x92 0x9E 0x7B 0x7D 0x37 0x33 0x38 0xD4 0x1C 0xBB 0x29 0xE7 0x7E 0xF6 0xEA 0x29 0x79 0xCF 0x0E 0xC2 0x79 0xD5 0xE1 0x9B 0x00 0xE5 0x82 0xD0 0x69 0x2C 0x98 0x12 0x2C 0x0F 0x72 0x2B 0x36 0x2C 0xC5 0xA3 0x5A 0x09 0xBA 0x04 0x8D 0x01 0xC4 0xDF 0x1F 0x85 0x6A 0x6D 0xD4 0xF9 0x35 0x55 0xF5 0x35 0xA1 0x8C 0xB3 0x04 0x3B 0xBE 0x03 0x36 0x38 0x83 0x69 0xED 0x2A 0x9F 0x26 0xB6 0xB8 0x8A 0x7C 0xED 0x2B 0x17 0x16 0x9B 0xAF 0xD5 0xFC 0x97 0x2E 0x92 0x60 0x22 0x26 0xC6 0x77 0x8C 0x6E 0x48 0xF8 0xAE 0x30 0x48 0x79 0x02 0x38 0xA0 0x93 0xDB 0xCA 0x10 0x07 0x7E 0xD6 0x91 0xD4 0xC2 0xF5 0x89 0x5F 0x3B 0x08 0xB8 0xC6 0xCD 0xF7 0x83 0x80 0x8E 0xE3 0x10 0x54 0x1B 0x99 0x5C 0x69 0xD2 0x68 0x4B 0x68 0x6C 0x22 0x51 0xE8 0xCC 0x85 0x7E
Flags	REQUEST_BODY_PRESENT; NO_PARENT_FOUND
Response code	200
Content type	text/html
Parent URL	None

Figure 2.4: Examination of suspicious requests from a Zeus malware variant.

2.2 Our traffic analysis system architecture

The system we built to perform the analysis of the collected network traffic is depicted in figure 2.5. As one can see, benign and malicious captured files in the libpcap format are provided as input to the system. Those files are initially processed by Bro IDS which extracts inter alia the HTTP messages from them. Afterwards, the output of Bro IDS is given as input to Hviz which builds the request graph. Finally, the request graph is being analyzed by multiple scripts we have developed in order to find typical sequences of events occurring during human Web browsing and to compare them with those occurring on an infected host. The analysis is explained thoroughly in the following chapter.

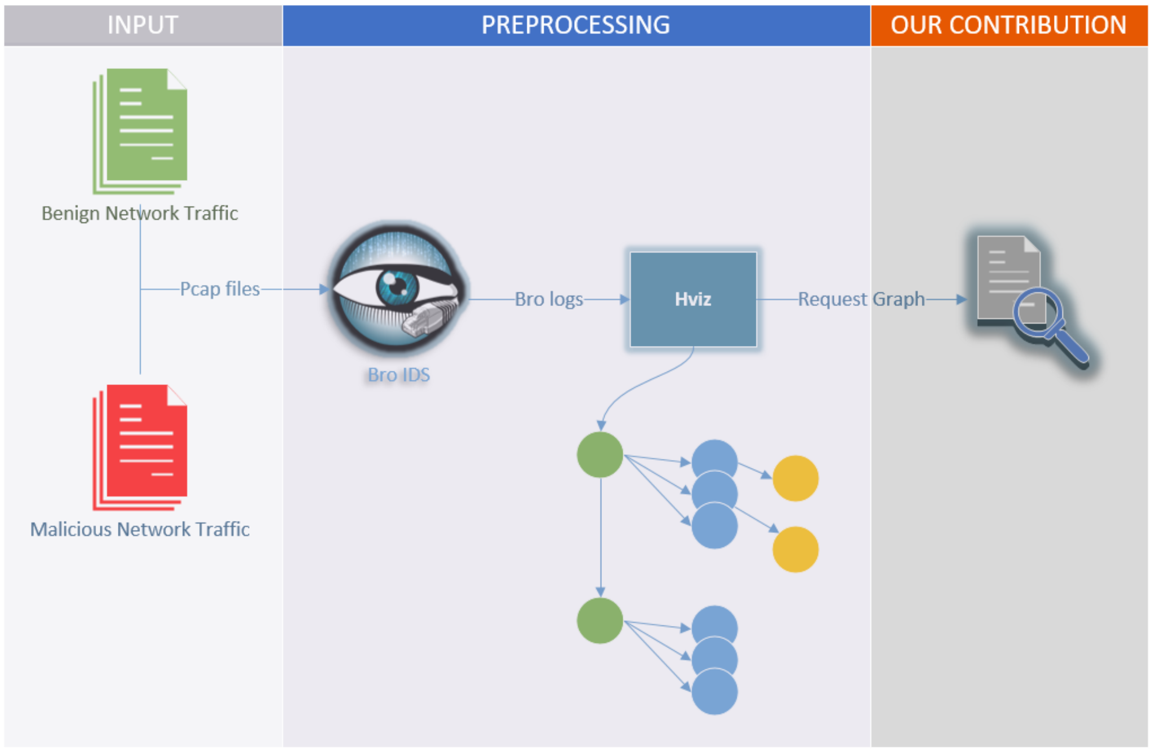


Figure 2.5: The process of analyzing a libpcap file.

Chapter 3

Baseline Analysis

Using the system introduced in section 2.2, we performed an extensive analysis of the benign as well as malicious traffic. Firstly, we describe how the dataset was obtained and subsequently, we present and explain the key findings of our analysis.

3.1 Datasets

Network traces were of key importance for the analysis and were needed to be provided as input to the system. Both benign and malicious network traces were collected from the following sources.

3.1.1 Benign traffic

Browsing simulation traffic

Both HTTP as well as SSL/TLS web traffic was recorded using the human-browsing simulation script described in section 2.1.1. More specifically, 10 browsing traces of approximately 40GB in total were recorded. For our purposes, an input list with the top 250 Alexa websites in Switzerland was used, after it was filtered to remove some websites with inappropriate content (e.g., adult content). The top ranked Alexa websites are calculated using a combination of average daily visitors and page views each month. The reason why this list was used, was to be as sure as possible that those websites would be benign. What is more, since they are used by many people, the daily web-browsing of most humans would be simulated.

ClickMiner traffic

ClickMiner [24] is a system which takes as input network traces and reconstructs interactions between the user and the browser. This system is different than ours, since it uses an instrumented browser to replay the recorded traffic in order to achieve the above goal. In order to evaluate their system, the authors conducted a user study involving 21 different participants. Each participant was requested to browse any website they wished for 20 minutes, given that they preserve their privacy (for example, they should not login to any site displaying any personal information, like Gmail, Facebook, e-banking sites, etc.) The authors of ClickMiner offer those full packet traces they collected during that study in their website (`http://clickminer.nis.cs.uga.edu`). From this source, 24 different browsing traces were accumulated.

3.1.2 Malicious traffic

Malicious pcap files were downloaded from the following sources.

- ***Contagiodump Blog*** [1]: Contagio is a collection of malware samples, threats, observations, and analyses. From there, an archive of malicious traces that other users have recorded and were willing to share was obtained.
- ***Malware-traffic-analysis Blog*** [4]: This blog contains a big collection of malware samples from 2013 till today as well as recorded traffic in the libpcap format from infected hosts. For each network trace it provides, there is an associated blog post with an analysis explaining among others the following:
  - How did the host get infected
  - Which were the associated malicious domains/files used
  - Which parts of the system were affected (e.g., registry values added)?
  - What requests did it perform to the outside world
  - How were data exfiltrated

This simplified the application of those traces considerably. After downloading files from both blogs, we filtered and kept the ones that contained HTTP traffic that we are focusing on this thesis. We ended up with approximately 500 malicious pcap files.

3.2 Some special malware C&C cases

In most of the collected malicious traces that we analyzed, we found the usual communication pattern between an infected host and the C&C server. More specifically:

- Perform an HTTP GET request to receive a command (polling the C&C) periodically
- Execute it and collect the results (if any)
- Send back the response in an obvious way (e.g., by appending various URL parameters to the request, either in plaintext or encoded).

However, there were some special cases that were found to be interesting and thus, worth mentioning:

- **TrojanCookies**: This malware was special because it communicates with the C&C server in an interesting way. The commands as well as the responses are encoded in the cookie and more specifically in the Set-Cookie HTTP header field. The variant analyzed used Base64 encoding, however, there are variants that additionally perform a single-byte XOR obfuscation. For example, decoding a corresponding Set-Cookie value gives the following string.

  ```
  command=GetCommand;clientkey=3954;hostname=victim;
  ```

  The client is polling the C&C server to receive a new command by providing a unique key as well as its hostname in order to be identified.
- **GCAL**: This malware uses Google Calendar to communicate with the C&C server using a hard-coded Google account. In order to get a command, the infected host downloads event feeds where each one of them contains one command from the attacker. After executing it, the compromised host posts back the results to the corresponding feed.

- **RSS Feeder** Victims of the RSS Feeder malware periodically perform web requests to RSS feeds in order to receive encrypted instructions/commands. Subsequently, after the execution of the command has finished, they send back the response by performing HTTP POST requests to another C&C server.

3.3 The Content-Type header field

According to the HTTP standard, the Content-Type [34] is a field in the HTTP headers that specifies the nature of the data in the body of an entity. Using this field, a user agent (e.g., the browser) can pick an appropriate mechanism to display the data to the user or deal with the them accordingly. As it will be shown in the following sections, the content-types of the HTTP responses played an important role in this thesis, thus, some results regarding them are presented here.

During the analysis of the content types of benign and malicious traces the following problem was encountered. Firstly, there exist different content types describing the same kind of data. For instance, to describe a javascript file, the most appropriate type (regarding the RFC) is *application/javascript*. However, other ones such as *text/javascript*, *application/x-javascript*, *text/js* etc. can be used, as well. Furthermore, since content types can be defined by each developer of the Web application, some careless mistakes (e.g., spelling) were noticed. Even though, most recent browsers are capable of understanding how the data needs to be displayed, in our analysis, in order to be as accurate as possible, those content types had to be merged and replaced with the proper ones.

Figures 3.1 and 3.2 depict the dominant content types in benign as well as malicious traffic, respectively. In benign traffic, the most used content-type relates to image files, since a regular webpage loads several of them to display its content. Moreover, GIF images are mainly used in advertisements and analytics services that are present in nearly all popular websites. For instance, Google Analytics [17] includes a script code block on web pages which references a Javascript file that is responsible for user tracking. The collected data are sent to the Analytics server via an HTTP GET request by requesting a single-pixel GIF image with a list of parameters attached.

On the contrary, malicious traces mostly contain PHP scripts which correspond to the C&C server communication whose content type usually is *text/html*. What is more, several Javascript objects can be found as they are mainly used to exploit the victim. Furthermore, it is shown that there are some *application/octet-stream* files which in this case are malicious binaries. In addition, there are several responses where the content-type is not set. This can happen unintentionally such as a 3xx redirect or intentionally by the malware. The reason that there are not only HTML content types as one would expect, but also image and CSS objects, is that some of the collected traces contain the infection phase of the victim, as described in section 1.4.

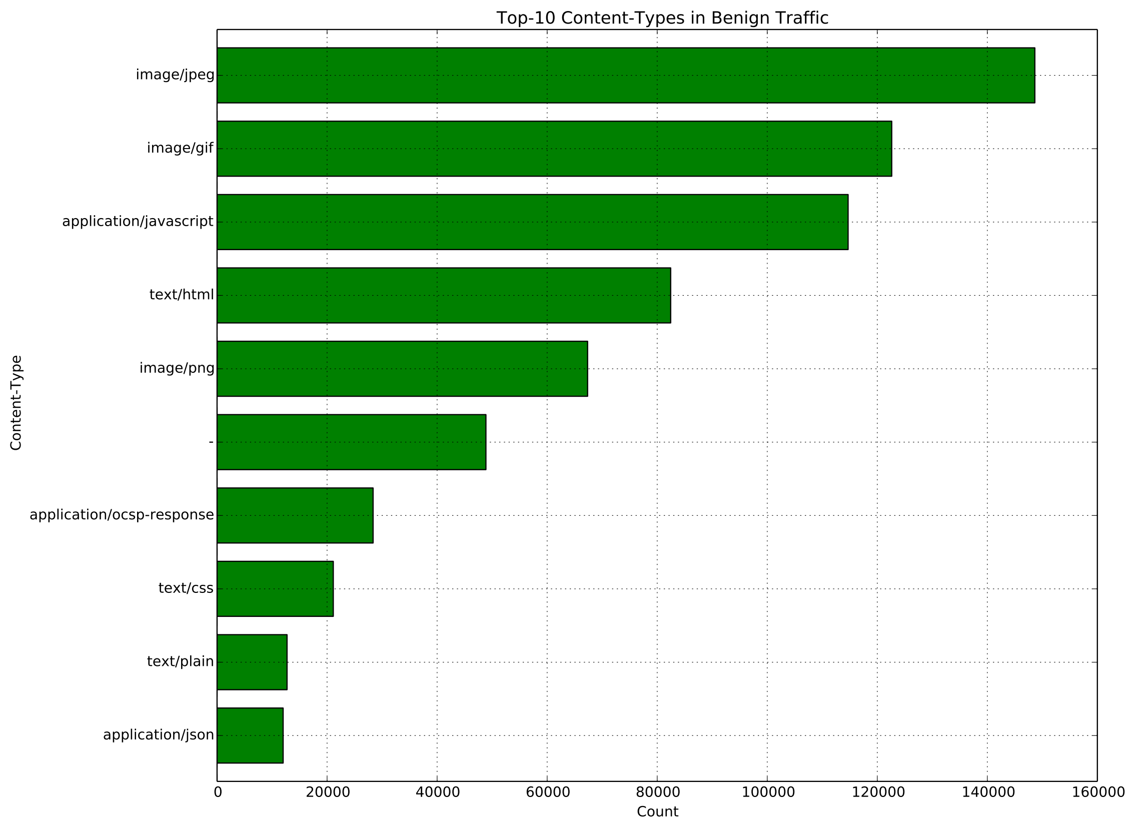


Figure 3.1: The top 10 content types found in benign traffic.

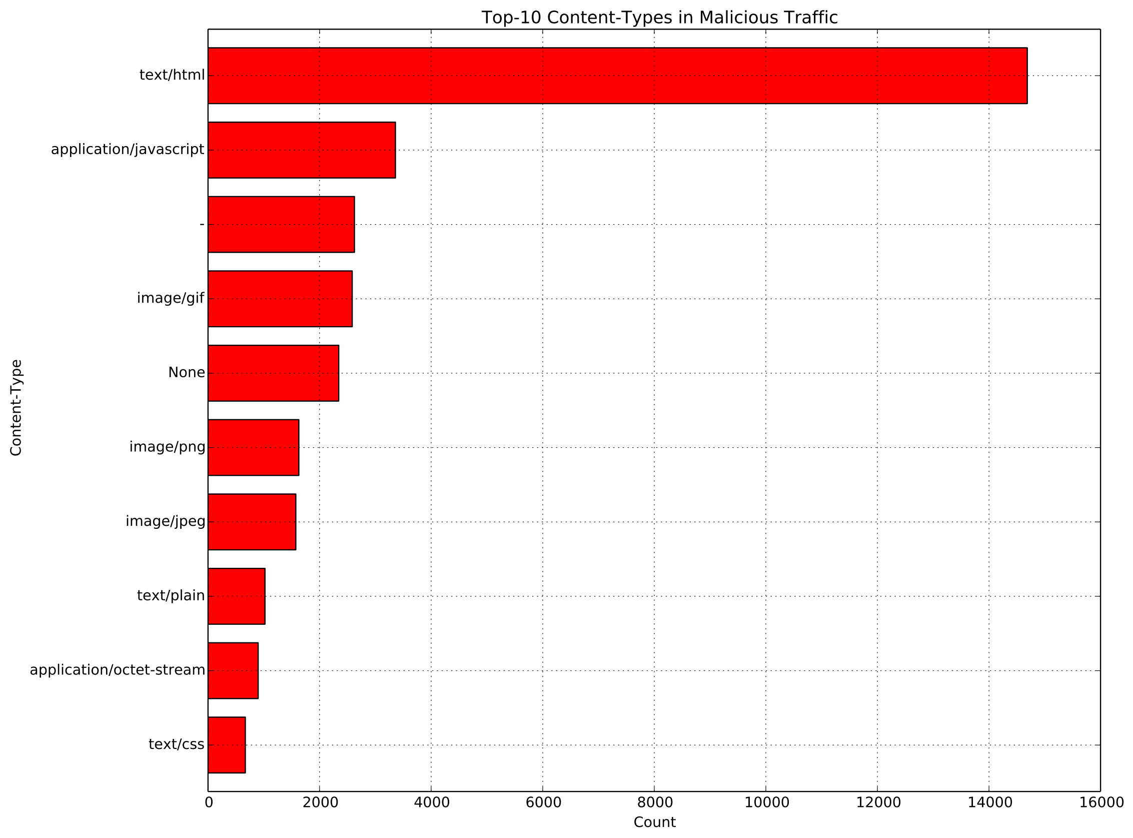


Figure 3.2: The top 10 content types found in malicious traffic.

3.4 Unrelated nodes

Analyzing the resulting request graph from Hviz, a node type with a special characteristic was discovered. This node did not have any parent (ancestors) or children (neighbors), and thus, it was named *unrelated node*. It turned out that this was a valuable finding which led to further analysis and interesting results.

3.4.1 Unrelated nodes in malicious traffic

Graphs that were output from malicious traces consisted in their majority of unrelated nodes. The reason why nodes extracted from malicious traces do not have neighbors is that a malware running on an infected host sends one request and receives one response periodically. In contrast to regular Web browsing, there are not any other requests that have to be performed since there is not any additional content that needs to be loaded from the C&C server (such as images, style-sheets, scripts etc.) Furthermore, the reason why these nodes do not have a parent is that most malware do not set the referer header when they perform an HTTP request since it is not necessary. Even if the referer was set to something benign (e.g., `https://www.google.com`), if the host has not visited that website before, there won't be any connection. Finally, inside malicious traffic, there are among others, requests to download other tools and malicious files, in order to perform fingerprinting in the network, to escalate privileges, to install a ransomware etc. Therefore, an investigator will see unrelated nodes in the time axis, representing requests to domains as the ones depicted in table 3.1.

This finding is important because it is a by-design characteristic of C&C communication and thus, it can be used to detect suspicious behavior. For instance, a malware like TrojanCookies described in section 3.2, even though it tries to conceal the traffic it is sending, can be very easily detected by displaying only the attributes of those unrelated nodes.

ftp.newaol.com	microupdate80.info	www.g1ikdcvns3sdsal.info
www.f5ds1jkkk4d.info	lpbmx.ru	macedonia.my1.ru
jrsx.jre.net.cn	squv.egozdq.com	sydmwk.5558x7.com
wqiwkb.wtcvxu.com	bsnf.bpfq02.com	ipkipk.3322.org

Table 3.1: Suspicious domain names from a variant of Sality malware.

3.4.2 Unrelated nodes in benign traffic

A simple detector to identify malicious nodes would be to mark every unrelated node as malicious and display its attributes to the investigator. Unfortunately, even though this would work with network traces that contain only malicious traffic or not that many requests (both benign and malicious), it would not in general. The reason is that one can find unrelated nodes in benign traffic, as well. Actually, it turned out that approximately 6.7% of the nodes in a benign network trace were unrelated. This percentage is not negligible. A graph can have a huge amount of nodes and in our case it had roughly 70000 nodes (see table 5.1.1). This means that a forensic investigator would have to go through approximately 4700 benign nodes in addition to the malicious ones which would be unaccepted.

3.4.3 Why unrelated nodes exist in benign traffic

As explained in section 2.1.5, in order for a node to have a parent, a valid HTTP referer header value needs to be present. However, this is not the case always for the following reasons.

- **The Online Certificate Status Protocol (OCSP)** [21]: The OCSP is an Internet protocol which is used as an alternative to certificate revocation lists (CRL). It allows applications to determine whether a digital certificate is valid. An OCSP client (e.g., the browser) issues a status request to the Certificate Authority (CA) and suspends acceptance of the certificate in question until it receives a response. Those requests/responses do not have a referer header set and do not cause any side requests. Thus, the nodes corresponding to them are unrelated. These requests can easily be identified by their content type which is *application/ocsp-response.*

- **Favicons**: As described in section 3.1, Firefox was used to collect the majority of the benign network traces. We found that whenever Firefox sends an HTTP request to retrieve the favicon of a website, it does not include the referer field in the HTTP request headers. As it turned out, this happens due to the *<link rel='icon' ....>* tag found in the HTML source code of web pages. There is a known bug associated with the above behavior [15] which has been resolved but not fixed yet. Performing a test with the latest version of Google Chrome, the same bug was not present.

- **Privacy**: There are cases where the referer header can affect the user's privacy. For instance, a URL might contain personal information in its query strings in case of a GET request. This was the case with Facebook in 2010 [22]. More specifically, advertisers could track a user who clicked on their advertisement since its user ID was inside the URL in the referer header. We found that this was the case with many advertiser and analytic services. Thus, security-aware developers filter out those information by providing only the hostname of the origin website or removing it entirely when performing requests to an external website. In order to assist developers and offer them the ability to control the referer header, the World Wide Web Consortium (W3C) has developed a new standard called the referer policy [35].

  Another case which belongs to the same category is the transition from an SSL/TLS resource to an HTTP (downgrade). To avoid leaking sensitive information, browsers "should not include a referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol" [20]. This could happen deep inside the request graph and might be complicated to detect. For instance, a real example from the collected benign traces is the following which happened with Mediamarkt's website, the popular electronic shop in Switzerland. A client that visits `http://fotoservice.mediamarkt.ch` is redirected to another page which among other objects, loads some CSS files. One of those stylesheets fetches other ones from an SSL enabled website which in turn retrieves some PNG images using HTTP (downgrade) requests. Consequently, in those last requests the referer missing.

- **OPTIONS HTTP method**: "This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval" [3]. Even though, this method is rarely used (in our benign traffic they account for the 0.07% of the total requests), Firefox as well as Internet Explorer do not set the referer header when performing an OPTIONS request, in contrast to Chrome. As a result, nodes that relate to this HTTP method end up being unrelated.

- **Software polling for updates**: The same way malware poll a C&C server for new commands, benign software poll for updates. Some examples are Window's and Mozilla's update services, a Twitter client which checks for new tweets, an antivirus software that updates its signatures etc. Since these are just individual requests and responses, they are represented as unrelated nodes too.

- **Invalid Referer**: The referer header can have an invalid value which means that it does not correspond to a request URI of any previous node in the graph. Further investigation showed that this was either a programmer's or a browser's bug.

- **Redirect Implementation**: There are several different ways for a user to be redirected from a source to a destination website. Firstly, one way is with a 302 HTTP status code combined with the Location value in the HTTP response headers. Another way, is by using a 200 HTTP status code and the Refresh header or an HTML meta tag. In addition, a user can be redirected using Javascript. Depending on the implementation of the redirection, there are different behaviors of browsers to either keep or suppress the referer.

Chapter 4

Building a Request Classifier

In the previous chapter the notion of the unrelated node was introduced. In section 4.1, we describe an algorithm to connect each of them inside the graph. Furthermore, in sections 4.2 and 4.3, we present the process of building our classifier that can efficiently distinguish between benign and malicious requests.

4.1 Adding the missing links in the graph

In order to make the request graph more complete as well as reduce the number of unrelated nodes in benign traffic, an algorithm had to be developed. This algorithm uses similar techniques as the one developed in [26], however their goal is different as explained in section 1.3. This algorithm takes as main input an unrelated node n and a graph G and outputs the most likely parent in the graph or null if the node does not seem to belong in it. The output size is not fixed and one should be able to output more than one possible parents. Based on our analysis of unrelated nodes in 3.4.3, we developed the following algorithm.

4.1.1 The algorithm in high-level

As it is depicted in figure 4.1, the HTTP requests of an unrelated node's possible parents should have happened in a time-window some seconds behind. Thus, the candidate nodes belong to this window whose size is provided as input to the algorithm, as well. For our purposes, the time-window used was 60 or 6 seconds depending on the heuristic (described below) used.

For each of those possible parent nodes, their response body attribute is examined. For example, the response body of an HTML document will contain the HTML source code of the web page. This means that we can examine all the href and src attributes inside it and most of the times know whether the node in question is the actual parent. This can provide us with valuable information because it can contain the whole request URI of the unrelated node and thus, the parent. Furthermore, it can contain a relative path (e.g., /wp-content/uploads/images/logo.png) which needs to be combined with the current path to create the whole request URI.

On the one hand, the main advantage of this method is its accuracy. On the other hand, since it needs the response bodies of many nodes, it increases the graph size and the processing time, as well. Taking this into account, only the response bodies for content types that have been found to have the most neighbors are retrieved.

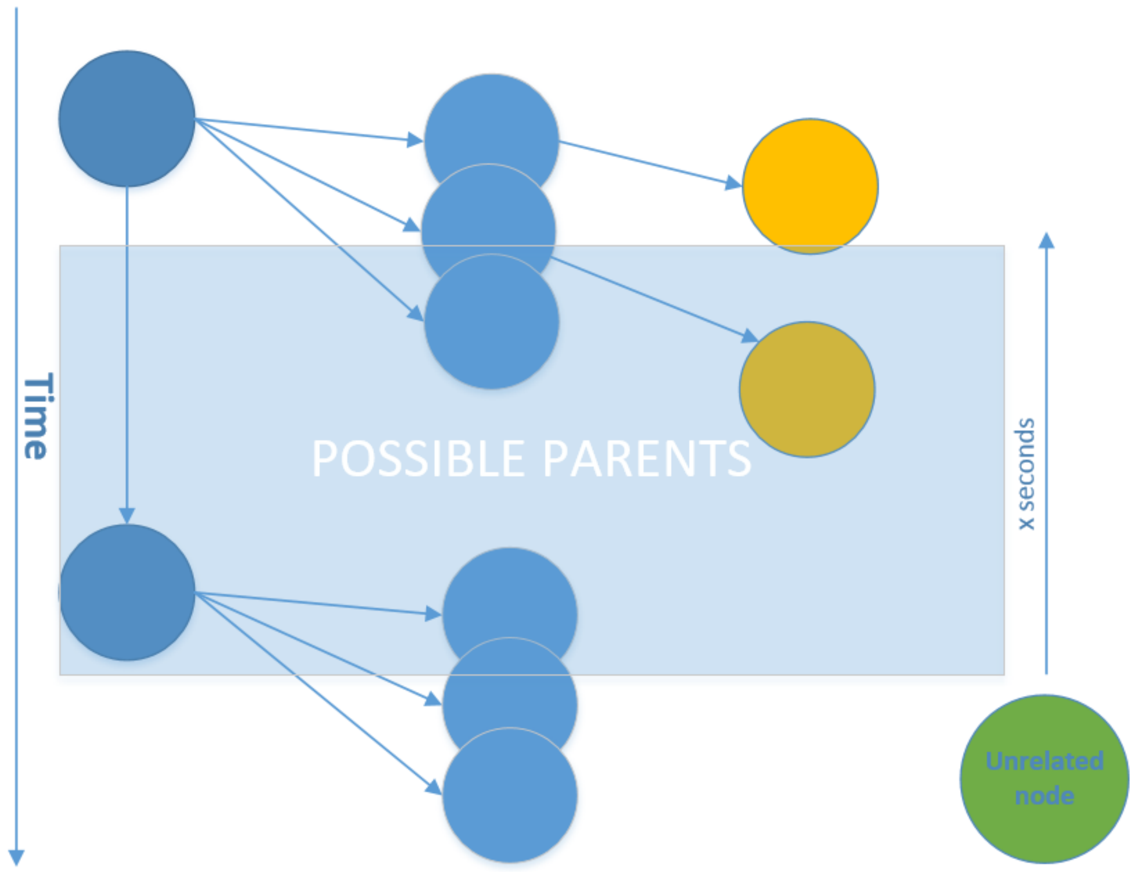


Figure 4.1: The time-window of an unrelated node's possible parents.

Regarding the favicons, even though the parents of a portion of them can be predicted using the above method, they can also be found in an easier way. For instance, if the unrelated node's request URI is `www.example.com/favicon.ico` then, the parent's should be `www.example.com`. By default the favicon is placed in the root directory of the web page and browsers know where to find it. However, it is a common practice that developers place their favicons in other directories. In that case, the algorithm is again able to find the parent using the domain name and the content type (*text/html*) of the possible parent nodes.

An important part of this algorithm is based on the nodes' content types for the following reasons. Firstly, there are certain content types which have more neighbors than others and also some of them that do not have any. For instance, an HTML document is more likely to perform more requests to load additional content (e.g., third party content) than a Javascript file. In addition to this, a node representing a request to a PNG image should not have any neighbors since it is not rational for this type of image to make additional requests. Moreover, the content type attribute of a node is relative to the kind of neighboring nodes it will have. For instance, a node whose content type is *text/html* is usual to have neighbors with content types *image/jpeg*, *text/css*, *application/javascript*, etc., whereas a *text/css* to have neighbors with *image/png*, *image/gif*, *application/font-woff* etc. content types. It does not make sense, however, after a *text/css* content type node, an application/octet-stream content type to be loaded. This is shown in figures 4.2 and 4.3.

Taking the above into consideration, the algorithm tries to match nodes of the same

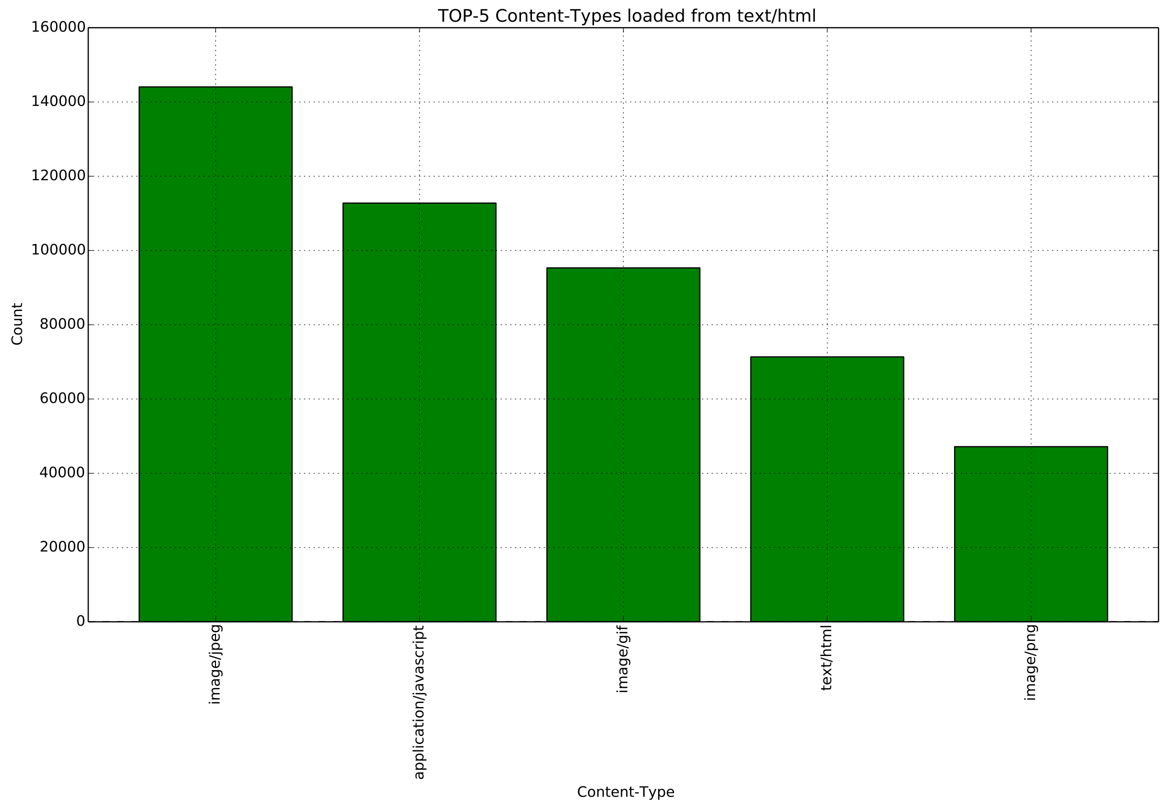


Figure 4.2: The top-5 Content-Types loaded from an HTML page.

content type to the same parent taking at the same time the hostname into account. In other words, an unrelated node is likely to be a neighbor of a node that already has as neighbors similar to it such as nodes of the same content type and domain. Furthermore, the algorithm takes as input the bi-grams of content types. Those were extracted by traversing all the graphs of the collected network traces and counting the top length-two sequences of the content types with most neighbors.

Finally, the algorithm uses a whitelist in order to filter out all those requests from benign software that can be running in the host. Even though, this list in our case is small as we did not have many of those requests, in a real case scenario one could adapt that list and add the known benign hostnames and/or IP addresses that their network is contacting.

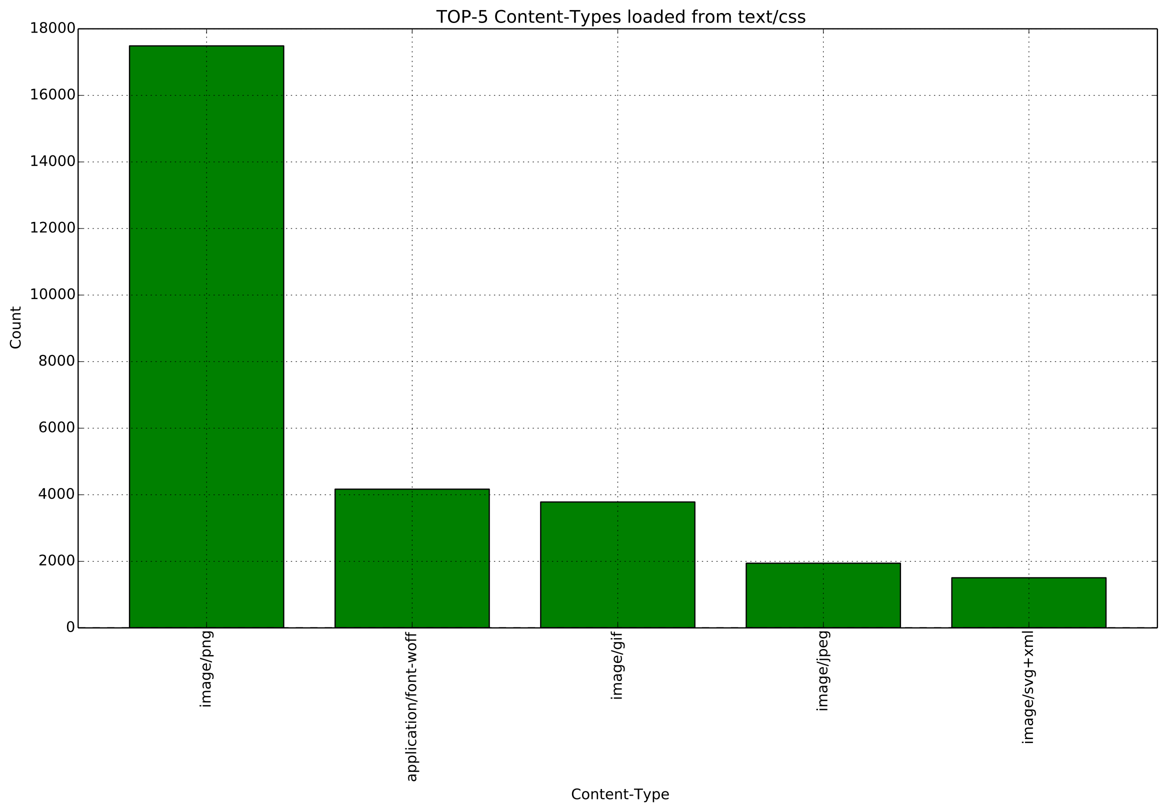


Figure 4.3: The top-5 Content-Types loaded from a CSS file.

More specifically, the algorithm for identifying a node *n's* parent can be described by the following steps.

1. Take as input the node in question *n*, the sorted by timestamp possible parents' lists *L* and *L'* in the time windows *tw1* and *tw2*, respectively and the sorted bigrams of the content types *BI*.
2. Check whether any of *n's* attributes such as its host, its content type etc. are whitelisted and if they are not go to step 3.
3. Keep all the nodes of *L* that are related, do not have the same URL as *n* and a content type relationship between parent and child node can be found in *BI*.
4. Traverse *L* backwards and examine each node's response body (if exists). First priority have the head nodes (user clicks), nodes with most neighbors and then, all the others. In case *n's* whole URL or relative is found inside the response body of a possible parent, return this parent.
5. If *n* corresponds to a favicon request then its parent is being searched by finding the initial head request to the same domain, as it was described in the example above.
6. Examine the nodes of *L'* and find the ones that have neighbors of the same host and the same content type as *n*. Sort them and return the most probable as the parent of *n*.

The time windows *tw1* and *tw2* used in the above algorithm were 60 and 6 seconds, respectively. The first time window, *tw1* relates to the maximum time gap between two sequential user clicks. The second relates to the maximum time that most of the embedded requests will be performed after a user-click. This number should be adjusted depending on the connection speed.

4.2 Simple classifier using only unrelated nodes

As explained in section 3.4.1, unrelated nodes in the request graph can be used to detect malicious traffic. This feature became more robust by the reduction of unrelated nodes in benign traffic which was achieved by adding the missing links in the graph. In figure 4.4 the pdf estimation of a boolean feature that shows if a node is unrelated is shown.
This feature can be used to separate the two classes (benign and malicious). Thus, a simple classifier using only this feature can be built in the following way. After executing the graph completion algorithm, every remaining unrelated node is marked as malicious and all the others as benign.

4.2.1 Evasion techniques

In this section we discuss some strategies that a malware could use to evade the above or similar classifiers so that it avoids detection. First and foremost, a malware can spoof the referer header of a malicious HTTP request to an innocuous website such as `https://www.google.com` as it was observed in our traces. This technique alone, however, is not adequate because there needs to be a request to that benign website before, in order for the parent node to be present. Thus, the malware has the following two options in order to forge a valid referer. Firstly, it can sniff the HTTP traffic and wait for the right moment to perform its malicious request while a user is browsing the Web. However, this might require privileges that the malware won't probably have. Secondly, it can issue a

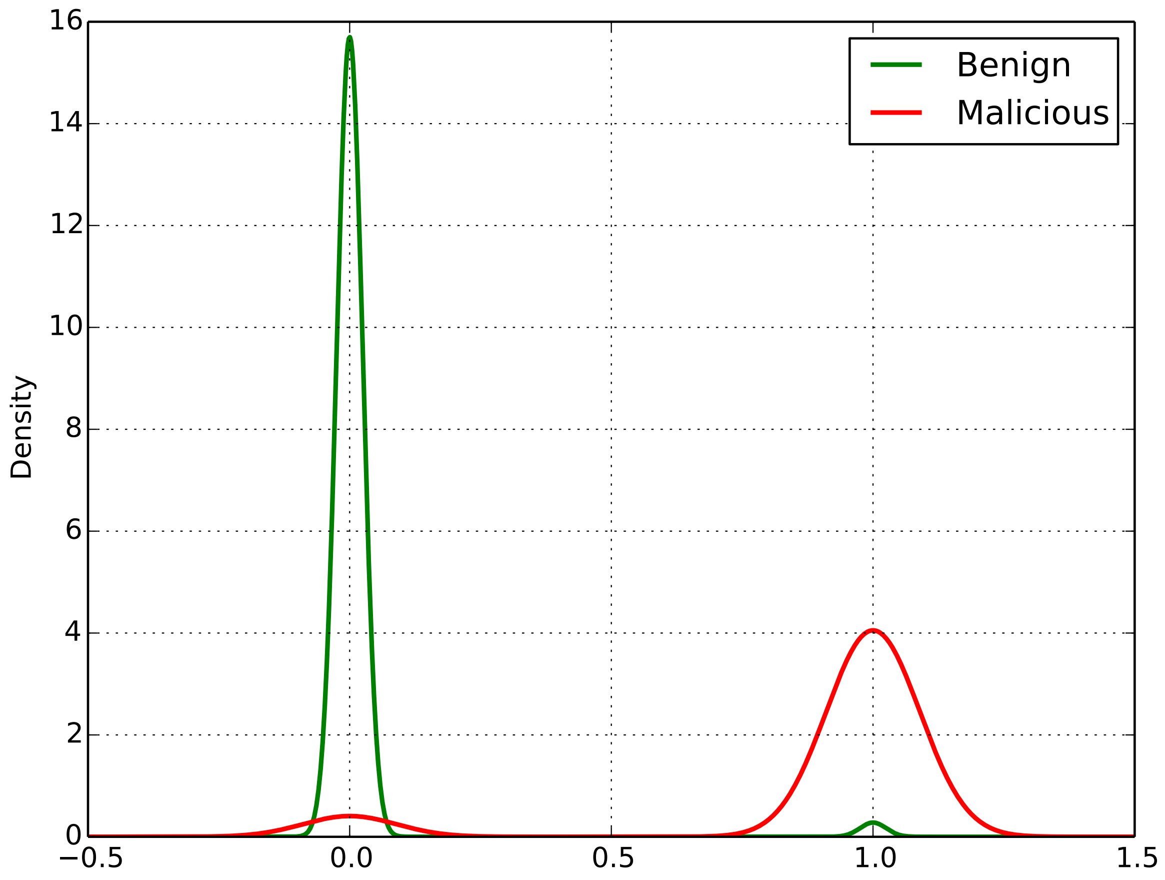


Figure 4.4: The estimated pdf of unrelated nodes in benign and malicious samples of the training set, after the execution of the graph completion algorithm.

request to a benign website and afterwards use this Web site's URL as referer.

Moreover, a malware could be controlled by abusing popular Web sites and using them as C&C servers. In particular, social network sites can be an effective C&C channel. Using a social network as control channel is achieved by the following simple steps. Firstly, the bot master posts malware instructions in a comment on a message board, and afterwards the malware visits the message board from the compromised machine, downloads the corresponding Web site, and extracts the instructions. A malware that wants to evade the above feature could partially load the social network's page with some of the embedded requests. This will imitate daily web browsing and will be challenging to detect.

4.3 Robust classifier

As discussed in section 4.2.1, there are multiple techniques for evading a simple classifier. However, each one of them increases the complexity of the malware and opens new windows for being detected. Still, it is preferable to build a more robust classifier which could detect obfuscated C&C traffic too. We describe a corresponding classifier in the following.

4.3.1 Feature selection

In order to make the classifier more robust, capable of detecting more sophisticated malware and at the same time make the malicious developers' task harder, more features have to be introduced. This way a malware will have to succeed in many different ”challenges” that each feature will pose.
Except from the unrelated node feature, multiple others were tried and evaluated. Here, the intuition behind the decision to use them or not is explained.

- **Connection Close**: We observed that some malicious requests set the Connection field in the request headers to Close instead of Keep-alive that every benign did. Whenever a request is sent and the value of this header is Close, the server receives it, sends back the response and closes the TCP socket. The client does the same when it receives the response, as well. As a result, the infected host's resources are not being wasted and thus, the malware remains stealthy. Moreover, the malware's infrastructure resources (e.g., a compromised Website) are consumed as little as possible.

- **Node's neighbors/predecessors**: In order to relax the condition of a node's neighbors and predecessors, those two features were introduced. It is rational that a malicious request does not have many children and any parent (unrelated node). However, in case of a possible evasion, those features could contribute in the detection, e.g., if fewer than the expected embedded requests occur. It is worth mentioning, that the number of predecessors (in our case either 0 or 1) was extracted after the execution of the graph completion algorithm.

- **Request's/response's size**: The intuition behind choosing those features is the following. Firstly, the length of the request relates to the data exfiltrated to the outside world or to the polling to the C&C server. Furthermore, the length of the response is relevant, as well. For instance, it can happen that the C&C does not reply at all or it just sends a small acknowledge back. Thus, we hypothesize that malicious samples will have longer requests and shorter responses.

- **Domain/URL**: It is known that malware use domain generation algorithms (DGA) in order to find one C&C server that is active. Thus, many C&C domain names can have some special characteristics they inherited from their generation algorithm. For example, a variant of Zeus trojan called Gameover Zeus uses between 23 and 28 characters (excluding the tld) domains [11]. Moreover, HTTP GET requests to a C&C server can be have a long URL with several parameters having sensitive data encoded in them. Consequently, the structure of the domain and the whole URL can play an important role in distinguishing between benign and malicious requests. We hypothesize that the malicious domains and URLs will be longer in malicious requests than in benign.

- **HTTP method**: The HTTP method used in combination with other features could be used. For instance, POST requests together with characteristics of the request body (e.g., if Base64 encoded data are present) could trigger an anomaly. However, this feature was not used since the collected benign traffic did not have enough POST requests. A reason is that the automated script did not submit any forms and thus, the appropriate ground truth for both classes does not exist in order to train a model.

- **Periodicity of C&C communication**: An infected host needs to communicate with the C&C server and this is most of the times done periodically. Many malware have either a fixed period or a randomly chosen waiting time between subsequent requests in a small interval. This by-design feature can be exploited in order to detect malware that can communicate stealthily, such as the ones using social networking Websites for C&C servers. This feature was not used in our case for the following reasons: many of the malicious traces contained only a few interactions with the C&C server and the rest were removed. What is more, each URL in benign traffic was visited only once because of the way the automated script worked. Even though to avoid overclaiming we did not use this feature, in a real case scenario it would probably be powerful.

- **User-Agent**: This feature can be really effective, since an unusual change in the User-Agent header field could indicate an anomaly. The user agent is usually hard-coded in the malware and can be used for authentication to the C&C server (e.g., the C&C responds only to requests from predefined user agents). None of the collected malicious traces used the same user agent as the one used to record the traffic. Therefore, this feature was not used in our classifiers, because it would have led to inaccurate better results (overclaim). However, in case one wanted to use this feature, they could first whitelist all user agents that are known to the specific network and notify an administrator whenever a change occurs.

Chapter 5

Evaluation

5.1 Evaluation of the graph completion algorithm

The goals of the algorithm are the following. First and foremost, it should reduce the number of unrelated nodes in a graph by adding them to the most likely parent. Moreover, it should not add links between unrelated nodes. In other words, it should not connect a foreign node such as a benign node extracted from a different network trace or a malicious one to the graph. Three kinds of evaluations were performed, as we will present in the following sections.

5.1.1 Measuring the ability to reduce the number of unrelated nodes

To test the ability of the algorithm to reduce the number of the unrelated nodes in a graph, we evaluated it with five randomly picked benign traces we have collected. Ideally, all of the unrelated nodes should be connected in the graph. The results are shown in table 5.1.

Trace ID	# of nodes	# of unrelated nodes before/after	Reduction factor
20160428	75306	5300/89	59
20160502	68489	4646/68	68
20160503	73413	5507/116	47
20160508am	68698	4220/135	31
20160508pm	63670	3832/87	44
Average	**69915**	**4701/138**	**50**

Table 5.1: The reduction of unrelated nodes by executing the algorithm on 5 random network traces.

From this table the following conclusions can be drawn. Each graph generated by a benign trace consists of approximately 70000 nodes, 6.7% of those are unrelated. After running the algorithm the number of unrelated nodes were reduced by a factor of 50.

5.1.2 Injection of unrelated nodes of benign traffic into benign traffic

The algorithm should not connect nodes from foreign graphs (false connections). Thus, for this evaluation, different traces of benign traffic had to be merged. More specifically, having a randomly picked benign trace's request graph, unrelated nodes of another (benign) trace were injected at random timestamps into it. To visualize this test one can see figure 5.1.

This evaluation is more an intermediate step (since it does not reflect reality) and was performed, in order to get the intuition to conduct the following evaluation.

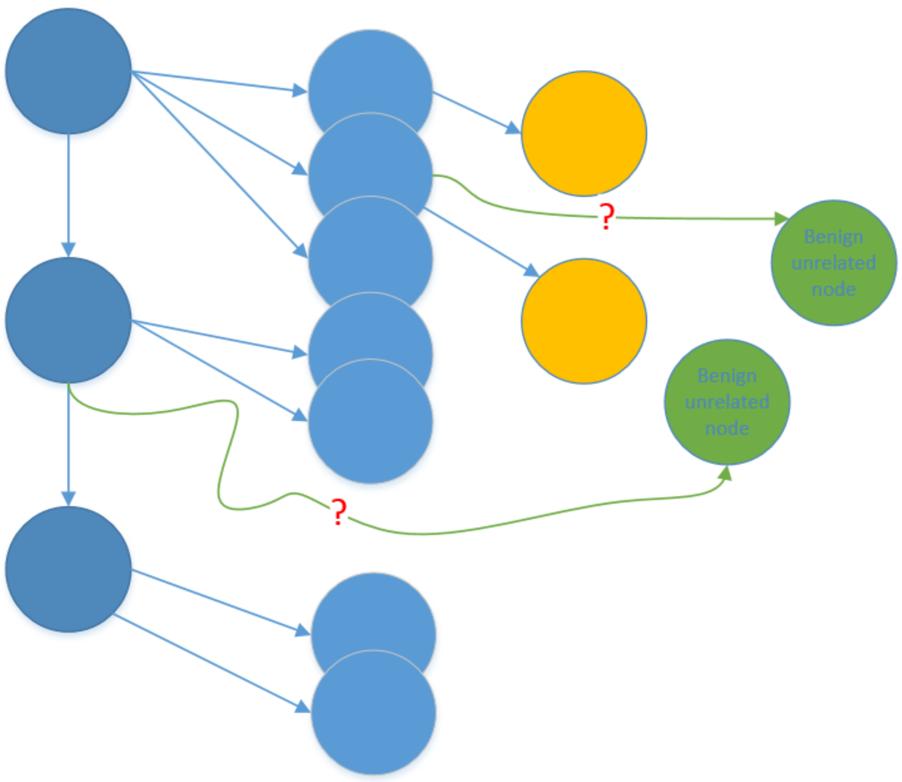


Figure 5.1: Evaluation setup - Injection of unrelated nodes of benign traffic into benign traffic.

The evaluation run five times and the results are provided in table 5.2. In each execution, the number of unrelated nodes injected was 3500, since this was close to the minimum that can be found inside a single benign network trace.

Injected From	Injected To	False Connections	False Connections (%)
20160428	20160429	45	1.2
20160502	20160516	53	1.5
20160508pm	20160507	48	1.3
20160428	20160508am	50	1.4
20160510	20160502	43	1.2
Average		**48**	**1.3**

Table 5.2: Injection of unrelated nodes of benign traffic into benign traffic.

1.3% of the nodes injected were misinterpreted and classified as part of the graph. The reasons why this happened are the following. Firstly, the benign traffic was recorded using the same list of websites (as described in section 3.1). Some of them might appear in both the injecting and injected trace which means that it could be the case they get connected. Furthermore, most of those websites use the same advertising and analytic services. For example, an unrelated node that corresponds to Google Analytics service can be injected in the right place (in terms of time) for another website using it too. Thus, this node will be identified as part of that traffic and falsely be linked to it. However, this traffic is benign and as a result, the effect of a misclassification like that is negligible.

5.1.3 Injection of unrelated nodes of malicious traffic into benign traffic

A similar to the above evaluation was performed, however, this time by merging benign with malicious traffic. More specifically, 3500 unrelated nodes from several malicious traces were injected into a randomly picked benign trace. Then, the algorithm was executed and the number of false connections was measured. Here it is worth mentioning that the traces from where the unrelated nodes were extracted did not only contain C&C traffic. This evaluation's results have more weight than the previous's, since they are closer to the task of detecting malware. That is an infected host which is browsing the Web and at the same time activity by malware running on the background. Each false connection (foreign unrelated node identified as part of the injected trace) would result in an additional false negative for a classifier that displays an alert for every unrelated node.

Our hypothesis is that there will only be few false connections as the malicious traffic and benign traffic traces that we have are uncorrelated in a sense that the domains contacted are different. Thus, we expect not to observe connections like the ones in the previous evaluation. The results can be seen in table 5.3.

Injected To	**False Connections**	**False Connections (%)**
20160428	4	0.1
20160429	4	0.1
20160502	3	0.1
20160503	4	0.1
20160507	3	0.1
Average	**3.6**	**0.1**

Table 5.3: Injection of unrelated nodes of malicious traffic into benign traffic.

Our hypothesis is confirmed by the results. Only 0.1% of the injected traffic was connected to the benign graph. However, by further investigation it was found that those nodes were actually not malicious. Some malware check the Internet connectivity of a compromised host by contacting benign websites. For instance, some variants of Zeus banking trojan contact `https://www.google.com` or `https://www.bing.com`. This kind of nodes were the only ones that were falsely connected inside the benign graph.

5.1.4 Discussion

The graph completion algorithm succeeds in reducing the number of benign unrelated nodes in the graph and at the same time it keeps the number of false connections low. The above results confirm that we can use it to improve the simple classifier described in section 4.2.

5.2 Evaluation of the classifiers

5.2.1 Evaluation Setup

As described in section 3.1, we collected both benign and malicious traffic from various sources. In order to be able to train a classifier and evaluate its performance, ground truth had to be created. The latter consists of both benign and malicious samples where each sample corresponds to an HTTP request and its response. In other words, each sample consists of features extracted from a node in the graph.

Regarding the benign samples, we randomly chose them from several different benign traces. As explained in section 3.1.1, those samples are assumed to be benign because they correspond to requests to the most popular websites in Switzerland and made inside a controlled, freshly installed virtual machine. In order to have as accurate results as possible, we had to create the training and test sets from different processes. Thus, we randomly extracted the samples for the training and testing set from the Browsing Simulation and the Clickminer [24] recorded traces, respectively.

We performed a manual analysis of the malicious samples, in order to label each node in the malicious traffic. The reason is that even though most of the traffic was indeed malicious, the post infection traffic (C&C communication) had to be identified since this was the one we were focusing on. In addition, some of the malicious traces contained benign traffic (e.g., polling for updates, check for internet connectivity, etc.) which we had to filter. We performed this time-consuming analysis on 65 malicious traces. Then, we used samples extracted from the graphs produced by 80% of those traces for training and we separated the remaining 20% entirely and kept them for testing.

Our target classes are benign and malicious. We define the following in the scope of the malicious class:

- **True Positives**: The number of malicious requests that were classified as malicious.
- **False Positives**: The number of benign requests classified as malicious.
- **False Negatives**: The number of malicious requests classified as benign.

The following metrics were used for the evaluation.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

We evaluated over the above metrics using 100 randomly chosen benign and 100 randomly chosen malicious requests.

5.2.2 Evaluation of the classifier using only unrelated nodes

We evaluated the simple classifier described in section 4.2 over the test set and found the following results. This classifier is effective and achieved 97% precision, 92% recall and 94% f1-score in our test set.

Precision	**Recall**	**F1-score**
0.97	0.92	0.94

Table 5.4: Evaluation of classifier using only unrelated nodes over the test set.

5.2.3 Evaluation of the robust classifiers

We evaluated various different classifiers in order to find the most suitable one. The optimal feature selection was derived using 5-fold cross-validation in the training set with the goal of maximizing the f1-score and is depicted in table 5.5. Different feature combinations were evaluated and the one with the highest f1-score was used. The f1 scoring function takes into account precision and recall which in this type of classifier are both important. This maximizes the number of true positives while minimizing the number of false positives and false negatives.

Classifier\Feature	**Unrelated node**	**Number of neighbors**	**Number of predecessors**	**Connection close**	**Request length**	**Response length**	**Domain length**	**URL length**
K-Nearest Neighbors	X			X		X		
Quadratic Discriminant Analysis		X	X	X	X	X	X	
Bernoulli Naive Bayes	X		X	X	X	X		
Adaboost			X	X	X		X	X
Gradient Boosting		X	X	X	X		X	
Random Forest			X	X	X		X	X
Decision Tree	X			X	X	X	X	

Table 5.5: Optimal feature selection which maximizes the f1-score.

The f1-score achieved by each classifier in the cross-validation sets using the optimal feature combination is shown in table 5.6.

Classifier	**F1-score**
K-Nearest Neighbors	0.95
Quadratic Discriminant Analysis	0.90
Bernoulli Naive Bayes	0.89
Adaboost	0.98
Gradient Boosting	0.97
Random Forest	0.98
Decision Tree	0.97

Table 5.6: Maximum average f1-score obtained by 5-fold cross-validation over the training set for both classes.

In Table 5.7, we present the evaluation of different classifiers over the test set. The results are close to the cross-validation estimations. Among all tested models, the Gradient Boosting classifier had the best performance in the test set with 99% precision, 97% recall and 98% f1-score. This makes it the most suitable robust classifier for distinguishing between benign and malicious requests in our case.

Classifier	**Precision**	**Recall**	**F1-score**
K-Nearest Neighbors	0.98	0.97	0.97
Quadratic Discriminant Analysis	0.99	0.92	0.95
Bernoulli Naive Bayes	0.97	0.97	0.97
Adaboost	1.0	0.83	0.91
Gradient Boosting	**0.99**	**0.97**	**0.98**
Random Forest	1.0	0.93	0.96
Decision Tree	1.0	0.95	0.97

Table 5.7: Evaluation of different classifiers over the test set.

Chapter 6

Conclusion

6.1 Summary

This thesis aimed to provide a solution to the problem of detecting malicious Web traffic (C&C communication). A traffic analysis system using existing tools has been built in order to analyze network traces and detect C&C traffic. Benign and malicious traces were collected and analyzed in extent and several common patterns in them have been discovered. An algorithm has been developed in order to fill in the missing links in the request graph which reduced the number of unrelated nodes in the benign traces by a factor of 50. Several features have been extracted from the analysis of the collected network traces. Different classifiers were evaluated in order to built a robust one which would be hard to evade. The Gradient Boosting classifier has been found to perform best at detecting C&C traffic. It achieved a 99% precision, 97% recall and 98% f1-score.

6.2 Outlook

There are many future directions this work could follow. Firstly, there is room for improvement of the graph completion algorithm. More heuristics can be found by analyzing the benign nodes that were not connected to the graph and be added into it. What is more, the algorithm could be evaluated over logs recorded by different browsers and be adapted accordingly to each one of them.

The developed algorithm is focused on adding the missing links of unrelated nodes in the graph and is probably not suitable for connecting other than unrelated nodes. Thus, it would probably be challenging to alter it so that it has the correct output for any kind of node in the graph. In other words, given only a list of nodes that belong to the same network trace, building the request graph without using the referer header field assuming that user clicks happen sequentially. For instance, one could rely on the timestamps of the requests and the time gap between them, the knowledge of the number of requests each content type performs approximately, on a mapping of domain names to providers (e.g., analytics, cdn, advertisement) etc. In addition, one can use the heuristics provided in this work such as the response body attribute of each node. An algorithm which addresses this problem would be helpful since as it was stated, the referer header field can sometimes not be an accurate metric to use to build the request graph (e.g., privacy addons in the browser that remove the referer header, security-aware developers that suppress it).

Furthermore, the robust classifier can still be improved. Different features can be found and evaluated. It can be upgraded in order to support additional classes such as the infection phase traffic, the data exfiltration, the malicious file download etc. However, this would require a more extensive dataset to be created or obtained. In addition, an evaluation of the classifier using traces from a corporate network would confirm the hypothesis that this system is ready to be deployed.

Finally, the analysis system's output (e.g., the request graph and the HTML file containing all the requests) could be integrated in a web service where investigators would be able to upload their network traces and receive the output. This would allow to evaluate how the system behaves with different kinds of recorded traces and measure its efficiency. In addition, it could help improve it since clients of the service could contribute with feedback and interesting suggestions.

Bibliography

[1] Contagiodump Blog. http://contagiodump.blogspot.ch/. [Online; accessed August-2016].

[2] Exploit Kits: Past, Present and Future. http://sjc1-te-ftp.trendmicro.com/images/tex/graphs/exploit-kit-attack-scenario.jpg. [Online; accessed August-2016].

[3] HTTP Method Definitions. https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. [Online; accessed August-2016].

[4] Malware-Traffic-Analysis Blog. http://www.malware-traffic-analysis.net/. [Online; accessed August-2016].

[5] Mitmproxy-Mitmdump. https://mitmproxy.org/. [Online; accessed August-2016].

[6] Networkx. https://networkx.github.io/. [Online; accessed August-2016].

[7] Offensive DNS-Server. https://github.com/LeonardoNve/dns2proxy. [Online; accessed August-2016].

[8] Python Data Analysis Library. http://pandas.pydata.org/. [Online; accessed August-2016].

[9] SSLSplit. https://www.roe.ch/SSLsplit. [Online; accessed August-2016].

[10] The Bro Network Security Monitor. https://www.bro.org/. [Online; accessed August-2016].

[11] The DGA of NewGoz. https://johannesbader.ch/2014/12/the-dga-of-newgoz/. [Online; accessed August-2016].

[12] VirusTotal. https://www.virustotal.com/. [Online; accessed August-2016].

[13] Wireshark. https://www.wireshark.org/. [Online; accessed August-2016].

[14] BBC News. Security firm RSA offers to replace SecurID tokens. http://www.bbc.com/news/technology-13681566y. [Online; accessed August-2016].

[15] Bugzilla. Bug 1282878. https://bugzilla.mozilla.org/show_bug.cgi?id=1282878. [Online; accessed August-2016].

[16] P. Burghouwt, M. Spruit, and H. Sips. *Towards Detection of Botnet Communication through Social Media by Monitoring User Activity*, pages 131–143. Springer Berlin Heidelberg, 2011.

[17] Google Analytics. Tracking Code Overview. https://developers.google.com/analytics/resources/concepts/gaConceptsTrackingOverview. [Online; accessed August-2016].

[18] D. Gugelmann, F. Gasser, B. Ager, and V. Lenders. Hviz: Http(s) traffic aggregation and visualization for network forensics. *Digital Investigation*, Mar 2015.

[19] IETF. Http strict transport security (hsts). `https://tools.ietf.org/html/rfc6797`. [Online; accessed August-2016].

[20] IETF. Hypertext Transfer Protocol – HTTP/1.1. `https://tools.ietf.org/html/rfc2616`. [Online; accessed August-2016].

[21] IETF. Online Certificate Status Protocol - OCSP. `https://tools.ietf.org/html/rfc6960`. [Online; accessed August-2016].

[22] M. Jones. Protecting privacy with referrers. `https://www.facebook.com/notes/facebook-engineering/protecting-privacy-with-referrers/392382738919/`, 2010. [Online; accessed August-2016].

[23] S.-J. Kim, S. Lee, and B. Bae. Has-analyzer: Detecting http-based c&c based on the analysis of http activity sets. *TIIS*, 8(5):1801–1816, 2014.

[24] C. Neasbitt, R. Perdisci, K. Li, and T. Nelms. Clickminer: Towards forensic reconstruction of user-browser interactions from network traces. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1244–1255, New York, NY, USA, 2014. ACM.

[25] T. Nelms, R. Perdisci, and M. Ahamad. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *In Proc. 22nd USENIX Security Symposium (USENIX Security 13)*, pages 589–604, Washington, D.C., 2013. USENIX.

[26] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *Proc. 24th USENIX Security Symposium (USENIX Security 15)*, pages 1025–1040, Washington, D.C., Aug. 2015. USENIX Association.

[27] Reporting and Analysis Centre for Information Assurance MELANI. Technical report about the malware used in the cyberespionage against ruag. `https://www.melani.admin.ch/melani/en/home/dokumentation/reports/technical-reports/technical-report_apt_case_ruag.html`, 2016. [Online; accessed July-2016].

[28] RSA FraudAction Research Labs. Anatomy of an attack. `http://blogs.rsa.com/anatomy-of-an-attack`, 2011. [Online; accessed August-2016].

[29] RSA-Security. `https://www.rsa.com/en-us/company`.

[30] RUAG. `http://www.ruag.com/group/about-us/`.

[31] SeleniumHQ. `http://www.seleniumhq.org/`. [Online; accessed August-2016].

[32] SSLStrip. `https://moxie.org/software/sslstrip/`. [Online; accessed August-2016].

[33] Tcpdump/Libpcap. `http://www.tcpdump.org/`. [Online; accessed August-2016].

[34] W3. The Content-Type Header Field. `https://www.w3.org/Protocols/rfc1341/4_Content-Type.html`. [Online; accessed August-2016].

[35] W3C. Referer Policy. `https://w3c.github.io/webappsec-referrer-policy/`. [Online; accessed August-2016].