Combining Multiple Malware Detection Approaches for Achieving Higher Accuracy

Master's thesis

University of Twente

Author:
Jarmo (J.M.) van Lenthe

Graduation committee members:
Prof. dr. ir. Aiko Pras
dr. Anna Sperotto
Rick Hofstede M.Sc.
Jair Santanna M.Sc.

January 23, 2014

As malware poses a major threat on the Internet, malware detection and mitigation approaches have been developed and used in the battle against malware. Some malware samples elude these approaches, while some benign software is marked malicious. Having looked at the state of the art in detection approaches, we have combined three, namely honeypots, DNS data analysis and flow data analysis. All three are widely used in corporate networks and can be exerted for detecting malware. By conducting experiments in which a workstation in a closed environment gets infected by malware samples, we have observed that a honeypot is not an effective approach for malware detection, because no malware tried to reach our honeypot. However, DNS data analysis and flow data analysis can be combined to achieve synergy, by providing more information about whether a workstation is infected by malware, leading to more informed decisions.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

Malware poses a major threat on the Internet [12]. Malware is defined as software that is created to do unwanted action on a computer, and includes worms, Trojan horses, viruses, and bots [43]. Detection and mitigation of malware is essential, and because of that, approaches for detecting it have been proposed [13, 12, 10]. Honeypots, DNS data analysis and flow data analysis are such approaches, which are widely used and can be exerted for detecting malware on networks [44, 20, 64]. This is because most malware will try to propagate itself to other systems or, in case of botnet malware, will try to download commands from a Command & Control (C&C) server.

Honeypots were originally created to learn the methods attackers use, but are now also used for catching and analysing malware [44]. They are a traditional tool in the ongoing defence against attackers and malware. DNS data analysis is used by network administrators to, for instance, list what websites are visited with a higher frequency than others, but can be exerted for malware detection [20, 77]. Patterns in amount of DNS replies over time exist in DNS data that can point to a botnet infection [59]. Flow data analysis was originally proposed to gain information about flows in a network, for instance for billing and maintenance purposes, and is standardized in the capacity of IPFIX [32]. It can be used to detect malware by marking certain characteristics in the network traffic caused by malware [64]. In general, each approach is applied to detect a specific set of malware types in a specific kind of dataset.

The effectiveness of an approach can be measured in terms of accuracy, which is the ratio of correct classified samples divided by all samples. The accuracy of multiple approaches may improve by letting them work together, creating synergy. Therefore, we intuitively believe that we can achieve a higher accuracy by combining approaches for detecting malware compared to the accuracy of the individual approaches.

In this research, we will combine existing approaches that are widely used by network administrators [70]. We will correlate information from honeypot data, DNS data and flow data analysis. We will run detection systems that generate this data in parallel in order to minimize the false positives and false negatives and thus achieve a higher accuracy. For example, when quasi-random domain names are queried, which can be observed using DNS data analysis, and the system subsequently connects to the corresponding IP address on unusual ports, which can be detected by flow data analysis, we have two reasons to mark the system as infected by malware. In this way, the certainty that the system is infected by malware increases.

The goal of this research is to investigate how the combination of multiple approaches of malware detection systems improves the accuracy. This gives us our main research question:

> How does combining multiple approaches of malware detection systems improve the malware detection accuracy?

To answer the main question, we will do a literature study and conduct experiments. This gives us the following preliminary research questions.

- What is the state of the art on identifying malware-infected systems with honeypots, DNS data analysis, and flow data analysis?

- What types of malware can be detected with the combined approaches?

For each dataset, we will study the state of the art of the existing approaches in Chapter 2, Chapter 3, and Chapter 4. In Chapter 5 and Chapter 6, we will conduct an experiment in which we will run malware samples on a closed environment, while collecting information from different detection approaches. This will give us data sets for each approach. In this way, we can make an unbiased conclusion. At last, we show that using multiple approaches of identifying malware-infected systems increases the accuracy of the malware detection approaches. In the literature study, we will focus on what types of malware the approach can detect, how it detects the malware, and how accurate the approach is. A malware classification is needed for this. This will come from existing research, such as Grégio *et al.* [24]. The experiment we conduct consists of running malware in a closed environment while gathering information from the different detection approaches. We will analyse the results on the basis of the analysis methodss described in the state of the art.

2 HONEYPOTS

In this section, we will first describe what honeypots are, which types exist and what their respective uses are in Section 2.1. We will then describe the state of the art of using honeypots for malware detection in Section 2.2.

2.1 BACKGROUND

Honeypots are vulnerable systems that are placed in a network to be compromised [67]. These vulnerabilities are present on purpose. Honeypot systems are always observed to learn from the methods that attackers use to compromise a system and what they do when they have succeeded. A honeypot can be compromised in two ways [81]. The first is when an attacker get into the honeypot. The other is when a piece of malware propagates itself over the Internet and places a copy of itself on the honeypot. The scope of this thesis excludes the first from this research, because we focus on detecting malware. Because honeypots have no production value, every connection to a honeypot has to be considered suspicious. This means that terms like false positives and false negatives are not applicable to honeypots [4]. A connection can be benign or malicious. If a honeypot is reached by accident, and no further action is taken against it, it is benign. Uploading a file to a honeypot however, is malicious. In classifying attacks or malware as benign or malicious, there can be false positives and negatives.

The most high-level classification of honeypots can be made on the basis of activity level and interaction level. This is shown schematically in Figure 1. Based on activity, we differentiate two types of honeypots: client honeypots and server honeypots [29]. Server honeypots are the traditional, passive honeypots that expose vulnerable services and wait for a connection to be made to them, reacting on an attack. Client honeypots are active honeypots, crawling the network or visiting URLs that may be a source of malware infections [36]. This definition contradicts the global honeypot definition, because this honeypot does not get compromised by an attacker, but rather compromises itself by downloading malware explicitly. The scope of this thesis is on server honeypots, because they enable us to detect malware activity in the network. Honeypots can be anomaly-based or signature-based. Anomaly-based means that it acts on everything that is out of the ordinary. Most honeypots are anomaly-based, as it is placed in a network to detect all kinds of attacks. Signature-based means that the

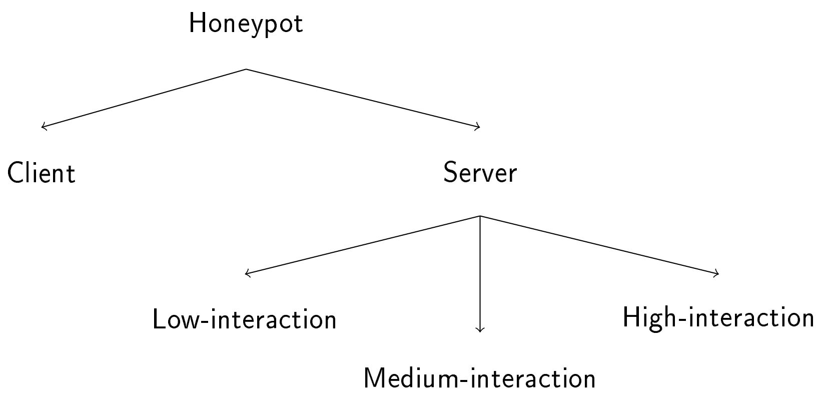

Figure 1: Classification of honeypots.

honeypot will only act when something happens that complies to a certain signature. When a honeypot is anomaly-based but performs analysis based on hashes (which is signature-based analysis), it cannot identify unknown malware, but it does catch it for later, manual, processing.

Server honeypots come in three different interaction levels: high, medium and low [42]. The interaction level is the level of interaction that the malware can have with the honeypot system. It brings in a trade-off between the need of monitoring the honeypot and the quality of the information that can be retrieved from the honeypot. A higher interaction level is more risky to get compromised, and must therefore be monitored more intensely, as compromised systems can be used to do damage to other systems. Low-interaction honeypots listen to a port and write everything that gets sent to it to a file, but do not need much monitoring. Medium-interaction honeypots are systems that run honeypot software packages which simulate services or vulnerabilities. Examples of these packages are *Kippo*[1], *Dionaea*[2] and *Glastopf*[3]. Instead of giving the attacker a full-fledged system with which they can interact, they simulate a normal system. The softwarer calculates an expected response and returns that to the attacker. Because medium-interaction honeypots interact with the attacker, more information is gathered about the attack, which brings risk, so the system must monitored more intense than low-interaction honeypots. High-interaction honeypots are full-fledged systems in which run normal services, so nothing is simulated. They offer the most information, when configured correctly, but need to be highly monitored, as the risk of exposing a complete system is highest.

1 https://code.google.com/p/kippo/
2 http://dionaea.carnivore.it/
3 http://glastopf.org/

Detection of malware by using honeypots has been an already widely investigated subject in the past years [81, 23, 63, 65]. The solutions proposed in literature differ greatly, in terms of how the analysis of malware samples found is done, whether one or more honeypots are used and the interaction levels of those honeypots. It describes proposals for medium-interaction and high-interaction honeypots. As low-interaction honeypots cannot interact with the attacker, they do not yield much information, and are therefore not described in literature. This section is divided per interaction level.

2.2.1 *Medium-interaction honeypots*

Most of the literature describes medium-interaction honeypots to detect malware. In Göbel [23], the honeypot software package *Amun*[4] is used to catch malware. *Amun* analyses all malware samples found with its Shellcode Analyzer. One of the first steps that are taken by the analyser, is looking through the uploaded malware code to find URLs. It is likely that new malware or instructions for the uploaded malware is located at those URLs. It will then download from these URLs. From the malware samples it gathers from there, *Amun* can make Snort rules. Snort[5] is a rule-based and host-based intrusion detection system. The fact that the Snort rules are created on the honeypot, makes that these rules are all correctly classifying intrusions, as there are no false positives. Of course, these rules must be very strict, in order to block as less benign traffic as possible.

Wichersky from Kaspersky Labs has researched how *mwcollect*[6], another medium-interaction honeypot packages functions when deployed on the Internet [78]. *Mwcollect* emulates multiple services and receives malware via those services. The malware gets run in *libemu*, a library which emulates shell code and responds with expected results, that is, results that would be yielded when issuing the same shell code on the real software package. *Mwcollect* monitors the behaviour of malware by detecting calls to the API of the operating system, such as Windows' `URLDownloadToFileA`. In that way, every connection to other systems can be detected.

Honeypots can work together in a network. This is called a *honeynet* [41]. They can be used to detect how malware behaves in a network. In Hassan *et al.* [28], multiple *Nepenthes*[7] honeypot software packages are deployed. The honeypots all send the data they capture to a central server. The central server parses all information and

---

4 http://amunhoney.sourceforge.net/
5 http://www.snort.org/
6 http://mwcollect.org
7 http://nepenthes.carnivore.it/

stores it in a database. With a Web site front-end to this database, statistics can be calculated from the information, such as a reputation list of IP addresses and a geo-location map of the origin of the attacks.

In Grégio *et al.* [24], a *distributed honeynet* of *honeyd* honeypots is deployed. *Honeyd* is a honeypot package that can emulate many vulnerabilities of many different services. A distributed honeynet means that the honeypots are in different networks. The *honeyd* honeypots do not process any data, but rather proxy all traffic on the open ports to *Nepenthes*, previously described, honeypots. The *Nepenthes* honeypots do the actual accepting and analysis of the malware. They have compared their solution with a single *Nepenthes* honeypot on the average downloads per day. The single honeypot downloaded 20 malware samples per day, while the distributed network downloaded 70 per day.

Adachi *et al.* [1] describe *BitSaucer*, which can generate a number of virtual honeypots on demand. *BitSaucer* uses *process-level virtualisation*, rather than *machine-level virtualisation*. In that way, more than 1000 virtual executions of a malware sample can take place on one machine. This allows *BitSaucer* to emulate a large network of systems on one system, which enables the created honeynet to observe malware behaviour in a network.

Musca *et al.* [44] have combined the medium-interaction honeypots *honeyd* and *metasploitable*. *Metasploitable* is an intentionally vulnerable Linux virtual machine that is primarily used for security training, testing of security tools, and practice penetration testing techniques [50]. Using the data of this honeynet, they are able to generate rules for the intrusion detection system *Snort*. This is an example of how honeypots may directly influence other systems, so that malware can be stopped more quickly.

Krueger *et al.* [34] use a *Web application honeypot* called *Glastopf*[8]. They have developed Automated, Semantics-aware Analysis of Payloads (*ASAP*), which is another approach of analysing malware, to work with the data from the honeypot. Krueger *et al.* [34] focus on three contributions of this *ASAP* framework. They extract an *alphabet of strings* from network payloads, which "concisely characterizes the network traffic by filtering out unnecessary protocol or volatile information via a multiple testing procedure and embeds the payloads into a vector space". This collection of *vector spaces* is then optimized using *matrix factorization*. This optimized matrix are used as basis for *communication templates*, which classifies and formats data from honeypots to make them clear for human interpretation. As said, they have applied this approach to network traffic captured by *Glastopf*. This honeypot was deployed for two months and collected an average of 3400 requests per day. From the requests that the honeypot has gathered, the researchers have used 1000 requests to val-

8 http://glastopf.org/

idate their proposition. From the traffic of these requests, *ASAP* has extracted communication templates on semantics of malware, vulnerabilities and attack sources. This part handles the detection of malware. *ASAP* can also be used for *malware communication analysis*. It can detect the HTTP component in the malware sample, so it detects Internet activity of a malware sample, such as where the malware gets its command from or where it can find its most recent version. *IRC components* get detected as well, so botnet malware that communicates over IRC can be found.

Malware is more and more becoming self-modifying, for it can then bypass anti-virus software [9]. To prevent this bypassing, Pauna proposed a self-adaptive honeypot system [51]. It is based on game theory and is able to detect rootkit malware [37]. Spitzner [66] described the adaptive honeypot as: "You simply plug it in and the honeypot does all the work for you. It automatically determines how many honeypots to deploy, how to deploy them, and what they should look like to blend in with your environment. Even better, the deployed honeypots change and adapt to your environment". The self-adaptive honeypot used is the Adaptive Honeypot Alternative (AHA). AHA may adopt behavioural strategies that can allow or block the execution of a program, substitute the program that will be executed or insult the attacker when he tries to issue a command, to irritate him so he will reveal his intentions.

Another honeynet is described by Szczepanik *et al.* [73]. When one honeypot gets infected by malware, another, identical but clean, honeypot checks what processes are running. By making a comparison of the running processes on the infected honeypot and the clean honeypot, processes that are started by the malware can be detected. This list is a helpful tool to analyse the behaviour of the malware.

A high-interaction honeypot system named *Jingu* is described in Chen *et al.* [11]. In that paper, *Jingu* is compared to the medium-interaction honeypot *honeyd*, a honeypot that simulates several known vulnerabilities. In two years of deployment, *Jingu* caught more than 500 intrusion events and 81 suspicious downloads. *Jingu* can be used to detect known exploits, but also *zero-day malware*, malware that is so new that there do not exist any signatures for it yet.

2.2.2 *High-interaction honeypots*

Another distributed honeynet can be found in Drozd *et al.* [18], who have combined *honeyd* honeypots with the high-interaction honeypot *Argos*[9] [54]. Although *Argos* is a software package, it is still a high-interaction honeypot, as it runs on a host machine with virtual machines that are the actual honeypot. Argos is based on *memory-tainting techniques*: the memory status of a clean honeypot is used

9 http://www.few.vu.nl/argos/

as starting point. All changed memory by the honeypot is marked tainted and should never be executed. Using memory-tainting, the researchers have detected malware that uses *buffer overflows*, an anomaly in a program in which a write action overruns the buffer's boundary and thus overwrites memory it should not access, causing the program's flow to be altered to the extend of the system being compromised. Drozd *et al.* have used a dataset similar to the NoAH project's dataset [46].

Kohlrausch [33] has used the dataset of the NoAH project. In his research, the detection and analysis of the W32.Conficker [60] worm by the use of the *Argos* honeypot is investigated. He followed the approach of the NoAH project. First, well-known attacks are performed, which are guaranteed to be recognized to establish a learning base set, from which workflows are calculated for less well-known attacks, the test set, which follow the well-known attacks.

Brunner *et al.* [8] have created *AWESOME*, the Automated Web Emulation for Secure Operation of a Malware-Analysis Environment. In *AWESOME*, medium-interaction and high-interaction honeypots can collaborate: novel attacks or malware samples are sent to the high-interaction honeypot, which is *Argos* in this research, while attacks and malware samples that have been seen before are sent to the medium-interaction honeypot. *Argos* runs in a virtual machine. The system on which it runs uses *virtual machine introspection* (*VMI*), pausing the execution of the VM to enable extraction and alteration of the program flow during runtime. Thus, all actions the malware performs can be monitored.

Srinivasan *et al.* [68] propose *Timescope*, a honeypot framework that is able to replay the infection of malware that has entered the machine on a virtual environment. By running the malware multiple times, and then investigating what aspects are overlapping, they find traces of what the malware caused and can exclude coincident changes.

2.2.3 *Conclusion*

From the literature described in this chapter, we conclude that for the automated execution of our experiment, we want to use a medium-interaction server honeypot. A client honeypot would not detect malware that is already on the network, but rather download and analyse new malware from the Internet. It must be medium-interaction, as the trade-off of being hacked and yielding useful information is best with medium-interaction honeypot for a corporate network. An additional advantage is that we don't have a full-fledged machine to be compromised, but only a robust program that we can still rely on after one infection. A further requirement is that the honeypot is anomaly-based, as we want to detect as many malware samples as we can from a remote honeypot system, and not only the ones that trigger a specific vulnerability. In Table 1, an overview of all methods described in this chapter can be found.

Table 1: Literature classification of detecting malware with honeypots.

Method	Package name	Single or multiple honeypots	Interaction level	Signature or anomaly-based	Analysis on
Göbel [23]	Amun	Single	Medium	Anomaly	Shellcode analysis
Hassan *et al.* [28]	Nepenthes	Multiple	Medium	Anomaly	MD5 hash
Wicherski[78]	Mwcollect	Single	Medium	Anomaly	libemu
Szczepanik *et al.* [73]	n/a	Multiple	Medium	Anomaly & signature	Process lists
Adachi *et al.* [1]	BitSaucer	Multiple	Medium	Anomaly	n/a
Grégio *et al.* [24]	honeyd & Nepenthes	Multiple	Medium	Anomaly	MD5 hash
Musca *et al.* [44]	honeyd & Metasploitable	Multiple	Medium	Anomaly	n/a
Krueger *et al.* [34]	Glastopf	Single	Medium	Anomaly	Web requests
Pauna [51]	AHA	Single	Medium	Anomaly	System calls
Brunner *et al.* [8]	AWESOME	Multiple	High & Medium	Anomaly	Memory-tainting
Chen *et al.* [11]	Jingu	Multiple	High	Signature	Shellcode analysis
Drozd *et al.* [18]	Argos	Multiple	High	Anomaly	Memory-tainting
Kohlraush [33]	Argos	Multiple	High	Anomaly	Memory-tainting
Srinivasan *et al.* [68]	Timescope	Single	High	Anomaly	System calls & shellcode analysis

3 DNS

In this section, we will describe what DNS is, how it works, why it is important to look at DNS data for malware detection in Section 3.1 and what the state of the art of the latter is in Section 3.2.

3.1 BACKGROUND

The Domain Name System (DNS) is a vital infrastructure within the Internet [15]. It is used to translate the more human-readable domain names to the corresponding computer-understandable IP address, as illustrated in Figure 2. A user wants to search on Google, so he types `google.com` in his browser. The browser doesn't know how to contact Google, because it only understands IP addresses. So the system first issues a DNS query to `google.com`. It sends this query to the primary DNS server that is configured in his operating system. Then there are two possibilities, the DNS knows the IP address of Google and sends it back to the system of the user, or it doesn't know Google's IP address. In that case, it will traverse the DNS server tree until it gets the IP address of Google authoritive DNS server, the server which knows the IP address of all domains ending in `google.com`. From this server, the user's primary DNS server will receive the IP address of `google.com` and sends it back to the user's system. The browser of the user's system can then browse `google.com`.

DNS data analysis allows network administrators to analyse traffic to external systems [16]. When internal systems try to resolve a domain name, they send a DNS request to the DNS server. The response of the server can be classified in two classes. One is a positive answer, an IP address to which the domain name resolves, for instance A, AAAA, and CNAME records. The other class is a negative answer, mostly NXDOMAIN responses [77], which means that the requested domain name is not registered at its namespace's registrar.

Botnet malware make extensive use of DNS [49]. As botnets are an increasing trend, with 25% of all online computers being part of a botnet in 2008 and 35% in 2010 [40], DNS data analysis is a possible detection approach for malware.

3.2 STATE OF THE ART

In the arms race of botnets between attackers and botnet detectors, the attackers are constantly developing new techniques to evade the

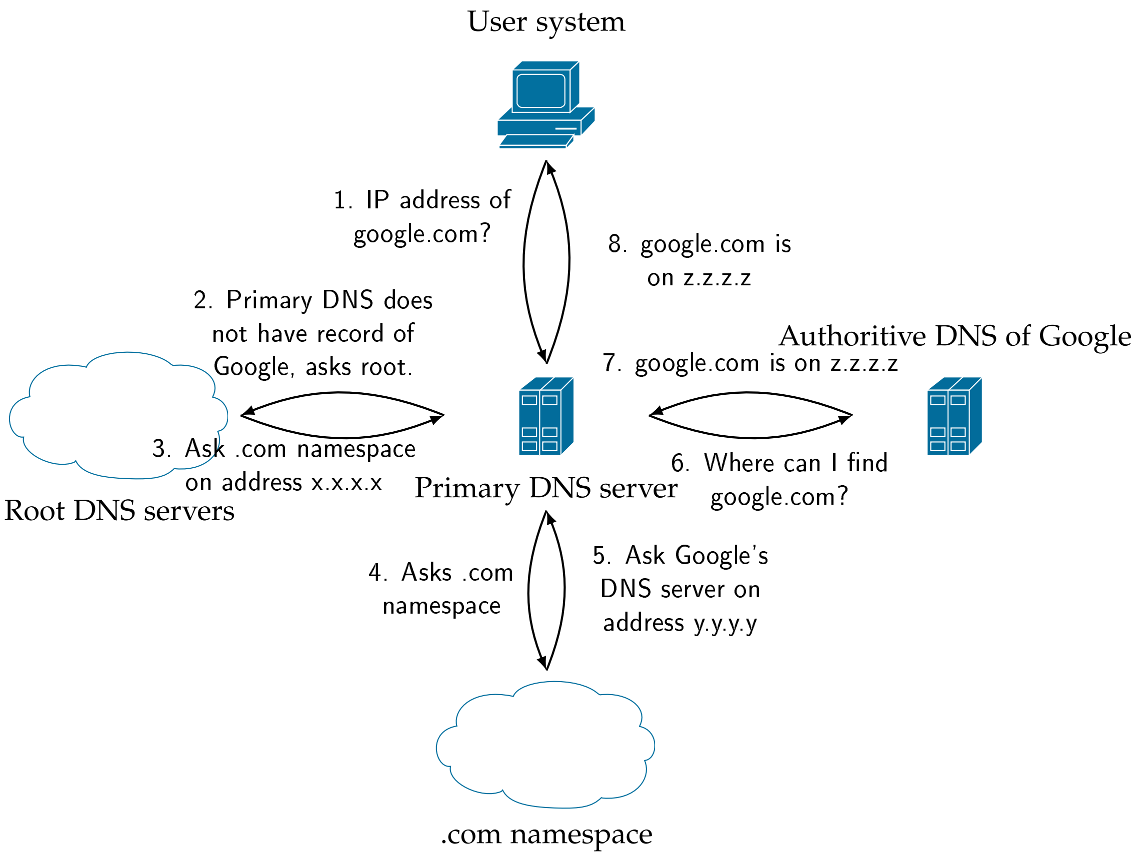

Figure 2: How DNS works: a system resolving google.com.

Table 2: Features to classify DNS records. Source: Bilge *et al.* [7]

Category	#	Feature
Time-based	1	Short life
	2	Daily similarity
	3	Repeating patterns
	4	Access ratio
DNS answer-based	5	Number of distinct IP addresses
	6	Number of distinct countries
	7	Number of domains share the IP address with
	8	Reverse DNS query results
TTL value-based	9	Average TTL
	10	Standard Deviation of TTL
	11	Number of distinct TTL values
	12	Number of TTL change
	13	Percentage usage of specific TTL ranges
Domain name-based	14	% of numerical characters
	15	% of the length of the Longest Meaningful Substring

detectors. In this section, we will investigate state of the art of using DNS data analysis for malware detection.

DNS traffic can be qualified on fifteen features, according to Bilge *et al.* [7] (see Table 2). They built *EXPOSURE*, a DNS data classifier. The fifteen features are categorised in four types, namely time-based features, DNS answer-based features, TTL value-based features and domain name-based features. Higher up in the DNS hierarchy, at the Top Level Domain DNS servers (such as the .com namespace from Figure 2) and Authoritative DNS servers, another system may detect malware-related domain names, namely *Kopis* [2]. This system makes use of the global visibility obtained from DNS traffic at the upper levels of the hierarchy and detects the malware-related domains based on several DNS resolution patterns.

What holds and must always hold, is that bots receive their commands from a Command & Control (C&C) server. In order to receive those, the bot must contact a C&C server periodically. If a C&C server is located at one IP address, the bot is easily turned into a zombie by blocking traffic to the C&C server's IP address from the infected system. Randomizing IP addresses is a hard task for attackers, as IP addresses are given out by ISPs from their pool, so the attacker cannot choose, and are hard to predict, especially when you need a lot of them. As an alternative, domain names can be used. When a C&C server is located at one domain name, it can be put on a blacklist and never be reached again [55]. Therefore, attackers have implemented Domain Generating Algorithms (DGAs) [49]. DGAs generate a list of domain names like in Table 3. Different DGAs generate domain names with different patterns. DGAs take a seed, like the first word of today's newspaper or, for instance, the current time to generate a different list every period of time [53]. Attackers and bots generate the same list of domain names. The attacker requires to register only one domain per period of time. The bot will try to connect to the C&C server by connecting to domains from the list. DNS requests for so many generated domains will result in NXDOMAIN responses, except for the domain that is registered. Detecting anomalous recurring NXDOMAIN reply rates is a way of using this technique to find bots in a network [59]. We refer to this method as the *NXDOMAIN method*. Botnets that use DGAs include: Bobax [71], Kraken [58], Sinowal (Torpig) [72], Srizbi [61], Conficker [52, 53], and Murofet [62]. Conficker-A, for instance, generates 250 domain names every three hours [53], of which only one has to be registered in that same period. The dissection of the DGA used by Conficker A [53], a specific type of the Conficker botnet malware, can be found in Listing 3. A methodology for algorithmically detecting DGA-generated domains is proposed by Yadav *et al.* [79], who use several statistical measures such as Kullback-Leibler divergence [35], Jaccard index [57], and Levenshtein edit distance [38]. This domain-fluxing, frequently changing the domain name on which the C&C server is located, which is investigated many times [3, 74, 72, 26, 79], and DGAs are used as a take-down evasion technique for botnets. Other malware can use DNS just as a normal computer user does, for instance to resolve a single domain name to signal an attacker that the infected system is compromised.

A measurement study on the *NXDOMAIN method* has been executed by Villamarín-Salomón *et al.* [76]. They have collected 11GB of DNS traffic data from the University of Pittsburgh. Almost all domain names that were found by studying abnormally high rates of NXDOMAIN responses, had been independently reported as suspicious by others.

Antanokakis *et al.* [3] have proposed a prototype called Pleiades for detecting bots in a network by passively processing DNS replies at the DNS server. When a cluster of NXDOMAIN requests is detected, it applies statistical learning techniques to build a model of the DGA. From this model, it can later detect systems that try to connect to the C&C server. The statistical learning techniques look whether the domain names have the same structure. Clients connecting to the DGA generated domains are suspect to be infected by bot malware.

Table 3: Example of domain names generated by a Domain Generating Algorithm (DGA). Source: Newman [49]

Domain name
mtizok-omik.ru
mpodod-axoz.ru
mdyhib-etop.ru
mbugaw-ewaq.ru
mkyqe-wukop.com
mfikyw-ybew.ru
mcali-fokaz.com
mbykyv-eceb.ru
mbavij-yris.ru
mmyqa-zezuv.com
mhapub-uluz.ru
mpibob-urok.ru
mrevoc-evyt.ru
msewo-xehem.com

Hao *et al.* [27] apply, with the NXDOMAIN technique in mind, the initial DNS behaviour after registration of a domain. From Domain Name Zone Alert systems, their system gets notified when a new domain is registered. From these domains, their system collects nameserver (NS), address (A), and mail server (MX) records. Their method focuses on botnets that are sending spam, but this technique can also be applied to other types of botnets, such as botnets that get instructions from a C&C server to initiate a Denial of Service (DoS) attack. From collected DNS records of the domains, their system looks at the distribution across IP address spaces, distribution across Asynchronous Systems (AS), in which the Internet divided, and the reputation of those ASes in light of hosting spam domains, and how much time passes before large amounts of queries are done to those DNS records. The theory is that legitimate domains are not as popular as spam domains after two days, but take more time. The theory of amounts of DNS queries over a period of time was also a part of the research done by Villamarín-Salomón *et al.* [76], but proved far less accurate than the *NXDOMAIN method* in that research.

In Choi *et al.* [14], DNS queries are examined and there is a track record for each domain name of how many hosts try to resolve that domain name per hour. 80% of the domains were visited by only one host per hour. The domains that were visited by more than 5 hosts per hour were only 7.5%. Within these domains, the greatest statistical similarity between domain names existed between domain

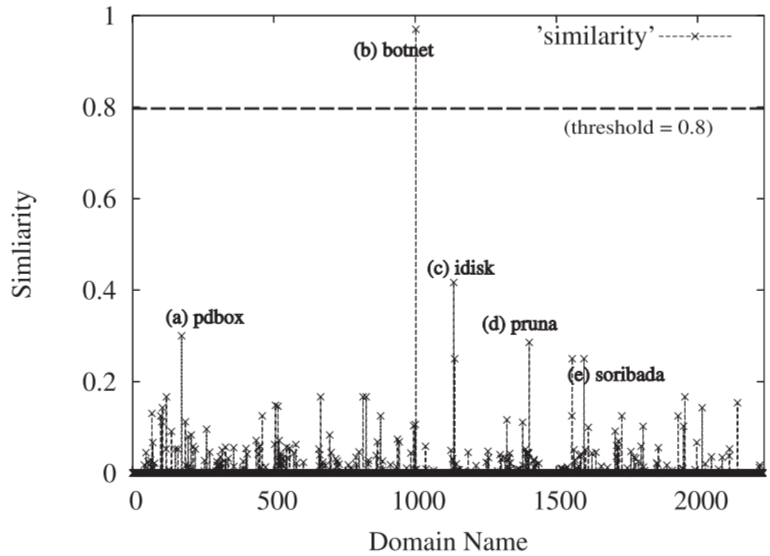

Figure 3: Statistical similarity between domain names is greatest with botnets. Source: Choi *et al.* [14]

names that are used by botnets, see Figure 3. This information can be used to correctly cluster multiple NXDOMAIN replies, as is done in the *NXDOMAIN method*.

3.3 CONCLUSION

From the literature described in this chapter, we have seen that the *NXDOMAIN method* is an effective malware detection method, which can be implemented in corporate networks without the need for extra machines. The features that are used with *EXPOSURE* can be used to classify the DNS requests that are observed.

4 FLOW DATA

In this section, we will investigate how malware can be detected with the use of flow data analysis, a technology for passive network measurements. We will describe in Section 4.1 how flow data is generated and how it can be analysed. In Section 4.2, we will discuss the state of the art in using flow data to detect malware.

4.1 BACKGROUND

A flow is a set of IP packets that pass through an observation point during a certain time interval [47]. A packet belongs to a flow if it satisfies all the defined properties of the flow, such as the packets all having the same source IP address or another set of . After being developed for network traffic accounting, usage for network forensics, and incident handling, flow data analysis is now also being used to discover malware [75]. Before flow data analysis, network traffic analysis was primarily done with packet analysis, which is still performed on specific types of network traffic, of which more details must be retained. Due to the large amounts of traffic that passes through networks today, this trends more and more to flow data [70]. Because flows are an aggregation of the traffic, it scales better to large networks. In addition, in many packet forwarding devices, Cisco's *NetFlow* [48], a flow export technology, is implemented. In order to export flow data on flow export supporting forwarding devices, *flow exporters*, it just requires to be configured in the device. Most corporate forwarding devices support flow data export. There is no need for extra forwarding devices or meters. This is another reason for trend towards the use of flow information. Flow exporters send the flow information to a *flow collector*, such as *nfcapd*, a part of the *nfdump* toolkit[1], which can be placed anywhere in the network. The flow collector receives all flow information, which is then available for all types of analysis, either manually or automatically. An illustrative explanation is shown in Figure 4.

Trivially, the flow data of a network contains more than the traffic information of just malware samples, but literature describes that malware-induced traffic has certain characteristics [64, 75], such as connecting to the same IP address, sending the same amount of bytes, every hour. By detecting those characteristics, malware-infected systems can be identified.

1 http://nfdump.sourceforge.net/

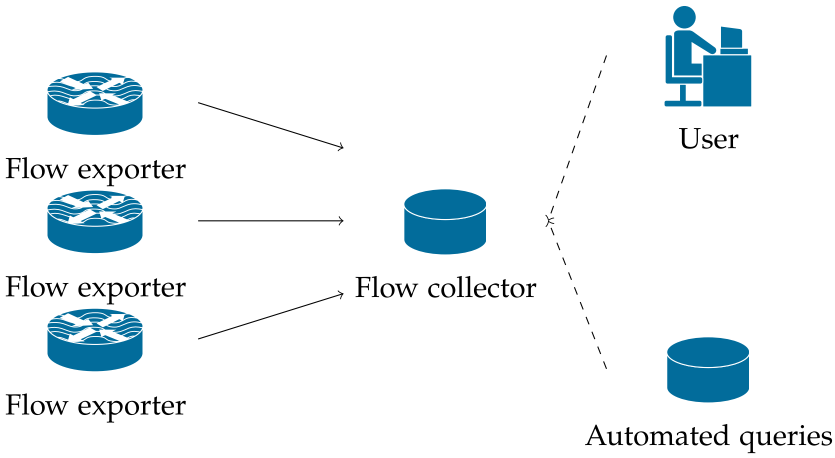

Figure 4: How flow data is exported, saved and queried.

4.2 STATE OF THE ART

The challenge with detecting malware on flow data is classifying certain traffic specifics are suspicious. Bilge *et al.* [6] have developed features for classifying flows, which are categorised as *flow size-based* features, *client access pattern-based* features, and *temporal* features, which are defined as follows. The flow size-based features indicate how many bytes are transferred. Flows that carry botnet commands have to be as small as possible in order to minimize their observable impact on the network. Flow sizes tend to not to vary greatly, because of the limited number of commands that are available in a C&C protocol. Conversely, flow sizes of benign servers tend to fluctuate greatly. With the client access patterns-based features, it is assumed that many bots run the same version of the malware. This makes the expectation that all the bots access the C&C server in the same manner very plausible. Benign servers are contacted in many different ways, due to human actions. Classification on the temporal features is based on the fact that bots try to contact the C&C server periodically and with relatively short intervals. Therefore, bots also try to make contact with the C&C server when normal client do not use the network a lot, for instance, at night. This classification system is what *Disclosure* [6] focuses on. Because flow data provides less information than a full packet capture, this approach could more likely contain false positives. They conclude that Disclosure can be tweaked to decrease the false positive rate to less than 0.5%, but in the large amounts of traffic of today, that is too much. Disclosure therefore includes a module to correlate data from other malware detection sources.

Berthier *et al.* [5] developed *Nfsight*, a tool which, apart from visualising traffic information, carries a heuristic-based intrusion detection and alerting system. The system was tested on 30 minutes of data from a border router of a university network. The information

Nfsight generates is structured with the use of rules, which are organized in three categories, namely malformed flows, one-to-many relationships and many-to-one relationships. The information is used to create communication structures, which are used to detect intrusions, but can also be applied in detecting peer-to-peer (P2P) or botnet malware. This classifying on the basis of one-to-many and many-to-one relationships relate to the client access pattern-based features proposed by Bilge *et al.*.

For the discovery of botnets, Gu *et al.* [25] have proposed *BotMiner*, which analyses network traffic via two monitors, one with flow data and one with the intrusion detection system *Snort*[2]. In the flow data monitor, flows from or to IP addresses of popular websites, such as Google of Facebook, are filtered, as well as traffic that only goes in one direction, because it is unlikely that contact with C&C servers behaves that way. For the remaining flows, the number of flows per hour, the number of packets per flow, the average number of bytes per packet, and the average number of bytes per second are calculated. Then a clustering of the flows is made, consisting of normal and suspicious flows. Gu *et al.* conclude that their framework can detect any kind of botnet, with very low false positive rates; a maximum of 0.3% was measured in their dataset. The classification features they use can be categorised as flow size and client access pattern-based features of Bilge *et al.*.

In Skrzewski [64], a system using flow count with regard to flow duration is proposed, and can therefore be grouped under the temporal features from Bilge *et al.*. An application makes several flows to the outside worlds. By counting the flows after settings several thresholds in the duration of the flows, differences prove to exist between infected and clean systems. Infected systems generate more flows that have a short duration.

Detection of P2P botnets using flow data is combined with using *PageRank*[3] in François *et al.* [21]. PageRank is Google's way to stating the relative importance of a website. It is based on two factors, the amount of links to the page on other pages, and the relative importance of the linking pages. They have experimented their method on three types of botnet topologies. The false positive rate in each of the experiments was 6% or less. As in Yen *et al.* [80], the hard part of marking clusters of systems as malicious is making the distinction between file-sharing P2P networks and P2P botnets, i.e. benign and malicious. The methodology for this is making distinctions on traffic volume, peer churn, and whether the network is human or machine driven.

2 http://snort.org
3 http://www.google.com/competition/howgooglesearchworks.html

4.3 CONCLUSION

From the literature discussed in this chapter, we have seen that there are many different features on which flows can be classified in order to mark them as originating from malware. The classification of Bilge *et al.* is the most detailed classification proposed to the best of our knowledge, which makes it an informative disquisition of flow characteristics.

5 EXPERIMENT SETUP

In this chapter, analysis of a honeypot, DNS data, and flow data are combined to achieve synergy in detecting malware. We will first describe the general setup of our experiment environment, after which we will explain the different parts of the setup more specific.

In order to analyse the accuracy of multiple malware detection approaches, we have set up a closed environment, which is illustrated in Figure 5. It consists of four machines, one host system with three Kernel Virtual Machine guests (KVM). The three KVM virtual machines are a honeypot, a DNS server and a workstation (a detailed description of our KVM structure can be found in Appendix D). The workstation will be infected by a total of 997 samples of malware, which is a collection of all available 64-bit executables malware samples for Windows put together on July 13, 2013 on VirusShare, which we downloaded on November 21, 2013. We chose 64-bit systems because 64-bit systems are a trend [45]. There are some of these malware sample repositories, such as malware.lu, frame4.net, offensivecomputing.net and virusshare.com, but we could only get an account at virusshare.com. At the date of accessing the VirusShare, the 21st of November, there were 14.5 million samples in the repository, which increases every day. A list of the malware samples we use can be found in Appendix A. In this section, we will first show the workflow of our experiment (Section 5.1). Second, we will explain the choices of data collection for the honeypot, DNS server and flow data (Section 5.2, Section 5.3, and Section 5.4), and lastly, we will explain the setup of the workstation (Section 5.5).

The host machine takes care of the networking. The host has a bridge device, which acts like a switch in normal network. The bridge can be connected to the physical network interface card of the host, providing the virtual machines with access to the Internet. During the preparation of the experiment, this connection is available. In this way, the honeypot and DNS server can access the Internet to download software. At the time of executing the malware, the connection to the Internet is switched off, to ensure that the system won't infect other systems on the network of the University of Twente. This limits our validation experiment, as the malware samples cannot connect to the servers to which it wants to connect, so we cannot get the same traffic characteristics. The other three systems are also connected to the bridge, resulting in a small network. This network setup resembles a corporate network, which is the reason that the system that is going to be infected is a workstation.

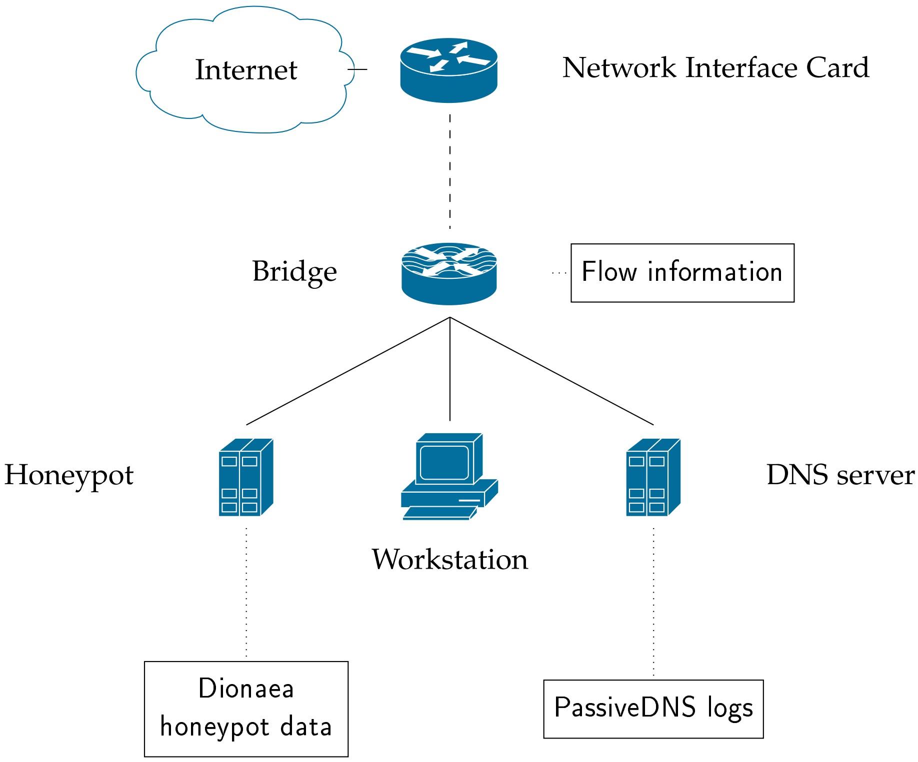

Figure 5: The network overview of our closed environment.

5.1 WORKFLOW

To generate a results set, the traffic characteristics of all malware samples, a script (see Appendix C) has been written to infect the workstation by running a piece of malware. It then waits for three minutes to allow the malware to infest the Windows workstation and the network. This should be enough time for malware to initialize itself, as malware tends to infest workstation in mere seconds [56]. In case of botnet malware, it should also be enough time to download commands from a C&C server. After this time, the script kills the workstation virtual machine and restores it to a snapshot of the pre-infected state. The process then repeats itself for the next sample. If the time of three minutes is not enough to yield viable results, we run the process again with the execution time of one hour. The script logs the timestamp it starts the infection of the workstation and the timestamp when the machine gets killed. These are used for matching data from the detection approaches later on. It is important that the clocks of the systems are synchronised for this to succeed, to match the timestamps from the script to that of the logs of the detection approaches. On our systems, this is not a problem, because the hardware clock of the physical machine is used in all systems. Restoring the workstation virtual machine is done in Logical Volume Manager (LVM). After restoring the snapshot, the workstation is booted again for the next infection. The LVM setup of our system can be found Appendix D.

5.2 HONEYPOT

The honeypot virtual machine runs a vanilla, pre-compiled Dionaea[1] package on Ubuntu[2]. As described in Chapter 2, Dionaea is a medium-interaction honeypot software package, a successor of Nepenthes and mwcollect, that is designed to collect malware. As a server-based honeypot, it waits for infected clients (or attackers) to connect to it, it does not visit malicious websites itself to see whether it can find malware, as that is what a client honeypot does. It runs the following services:

- FTP, port 21, used for file sharing;
- Samba, port 445, used for Samba file sharing and AD services;
- TFTP, port 69, used for file sharing;
- HTTP(S), port 80 & 443, used for serving Web pages;
- MSSQL, port 1433, used for MSSQL databases;
- MySQL, port 3306, used for MySQL databases; and
- SIP, port 5901, used for Internet telephony.

Dionaea can be classified as an anomaly-based honeypot, because it does not depend on a set of signatures. It therefore complies to our requirements set in Section 2.2.3. Dionaea can use the signature database of virustotal.com to provide extra information to the administrator by querying VirusTotal[3] with the MD5 hash of the malware sample, which is commonly used as an identification of the malware sample. Dionaea logs all connection and malware uploads in a sqlite database, and saves timestamps on every network interaction of the honeypot. These timestamps can be matched with the timestamps that are logged by the script, so we know which malware sample made which connection to the honeypot. In 2012, the European Network and Information Security Agency (ENISA) qualified Dionaea as an essential tool for Computer Emergency Response Teams [19].

5.3 DNS SERVER

The DNS server runs dnsmasq (a pre-compiled package for Debian), which is a DNS forwarder, which can have pre-configured DNS entries. By configuring the DNS server as the default DNS server on

1 http://dionaea.carnivore.it
2 http://www.ubuntu.org/
3 http://virustotal.com/

Listing 1: Example log rule created by *PassiveDNS*.

```
#timestamp||dns-client||dns-server||RR class||Query||
    Query Type||Answer||TTL||Count
1322849924.408856||10.1.1.1||8.8.8.8||IN||upload.
    youtube.com.||A||74.125.43.117||46587||5
```

the workstation, we ensure that all DNS queries that are done by the workstation which do not specify a DNS server themselves, are handled by our DNS server. To every DNS A query, the server responds that that domain name is associated with the IP address 1.2.3.4, rather than a NXDOMAIN. This ensures that the malware is convinced that the queried domain name is registered, so it will try to connect to the received IP address. On the DNS server, we run *PassiveDNS*[4], which analyses all traffic on the network adapter of the DNS server and logs every DNS reply that passes there, which in this case are the replies made by our dnsmasq. PassiveDNS creates logs rules like in Listing 1. It does not log the requests, as for every request, a reply is generated, which contains the request as well as the answers. In this way, we can investigate what domain names are queried. By also logging the timestamp, we can again match the reply to a specific malware sample. As the closed environment does not have access to the Internet, we cannot apply the *NXDOMAIN method* directly to the domain names that pass by the bridge. However, we can apply the *NXDOMAIN method* in retrospect to the logs generated by *PassiveDNS*. For example, as shown in Listing 1, 'ttupload.youtube.com is queried by 10.1.1.1 at server 8.8.8.8 and we see DNS server's answer that the domain name is associated with the IP address 74.125.43.117.

In order to obtain the domain names that were not queried at our DNS server, but rather by another DNS server of which the IP address was hardcoded in the malware, we have captured all packets that pass through the bridge with tcpdump in standard PCAP format. In real networks, collecting all DNS replies can be achieved by placing an additional *PassiveDNS* instance close to the border gateway, which we could not do, because we only have a switch, so no border gateway. In that way, DNS replies originating from external DNS servers are still passing through the system that runs *PassiveDNS*. By also running *PassiveDNS* on the internal DNS server, one can ensure not to miss any DNS replies.

4 http://github.com/gamelinux/passivedns

5.4 FLOW DATA

On the bridge in the host system, we export *NetFlow* data. We only use the source and destination IP addresses, ports, and the start time of the flow, the latter for matching the flows to the malware sample. To export the flows, we have used *nProbe*[5], a software flow exporter, in combination with *nfcapd*. *nProbe* sends the flow data to the specified collector. It runs *nfcapd* to receive the flow data and writes it to *nfdump*-readable files. There are more flows passing our bridge than from the workstation alone, such as flows from the honeypot, announcing its services, so we cannot match every flow to a malware sample, but we can look up the flows of the workstation during the period the malware sample was active. We have the start and stop time of the malware execution script in its log. An example result of a query we execute with *nfdump* is showed in Listing 2. In the example, eight flows are shown. The first six flows consist of DNS traffic. Our DNS server returned 1.2.3.4 as an DNS reply, as it does for all requests, which is observed as the last two flows from our workstation have that IP address as destination on port 1337.

5.5 WORKSTATION

The workstation is a Windows XP 64-bit machine, without any updates or service packs, as installing service packs is often delayed in corporate networks [22]. Since Q4 2012, Windows 7 is getting a larger market share than Windows XP [45], making it the most installed operating system today. However, the malware collection that we use contains mostly samples from the time that Windows XP was the most installed operating system, so we chose to work with Windows XP. By installing a SSH server (*WinSSHd*[6]) on this machine, we are able to run malware samples on it by issuing a command from the host machine.

5 http://www.ntop.org/products/nprobe/
6 http://www.bitvise.com/winsshd

Listing 2: Example result of a query executed with *nfdump*.

```
Date flow start Duration Proto Src IP Addr:Port Dst IP
   Addr:Port Packets Bytes Flows
2013-11-20 20:39:43.923 0.000 UDP 192.168.1.2:1033 ->
   192.168.1.3:53 1 67 1
2013-11-20 20:39:43.717 0.000 UDP 192.168.1.3:53 ->
   192.168.1.2:1033 1 83 1
2013-11-20 20:39:40.182 3.532 UDP 192.168.1.2:1029 ->
   192.168.1.3:53 2 134 1
2013-11-20 20:39:40.182 3.532 UDP 192.168.1.3:53 ->
   192.168.1.2:1029 2 166 1
2013-11-20 20:39:40.257 0.000 UDP 192.168.1.2:1030 ->
   192.168.1.3:53 1 67 1
2013-11-20 20:39:40.257 0.000 UDP 192.168.1.3:53 ->
   192.168.1.2:1030 1 83 1
2013-11-20 20:39:43.718 1.640 TCP 192.168.1.2:1032 ->
   1.2.3.4:1337 2 96 1
2013-11-20 20:39:40.258 3.047 TCP 192.168.1.2:1028 ->
   1.2.3.4:1337 2 96 1
Summary: total flows: 8, total bytes: 792, total
   packets: 12, avg bps: 1224, avg pps: 2, avg bpp: 66
Time window: 2013-11-20 20:38:43 - 2013-11-20 20:43:29
Total flows processed: 41, Blocks skipped: 0, Bytes
   read: 2160
Sys: 0.032s flows/second: 1281.2 Wall: 0.017s flows/
   second: 2314.6
```

6 EXPERIMENT RESULTS

In this chapter, we will show and discuss the results of our experiments. Firstly, we will show an overview of the aspects that we analyse on (Chapter 6). Secondly, we will explain the results per detection approach: honeypot (Section 6.1), DNS data (Section 6.2), and flow data (Section 6.3). Thirdly, we discuss the results of combining the approaches in Section 6.4. Finally, we show examples of samples that induced traffic which we did not expect (Section 6.5).

We analyse multiple aspects on which we can validate the results, which are derived from the propositions we have chosen from literature. We have aspects per detection approach and for the combined solution. An overview of the aspects is in Table 4. The general aspect will be analysed in this section, the approach-specific aspects in their respective sections.

Of all the 997 malware samples we have analysed, only 82 interacted with the network in the first three minutes after infection. As all network traffic is logged in the flow data, this is something we can easily obtain. Of the 82 samples that interacted, zero malware samples contacted our honeypot. 68 samples have queried at least one domain name. 50 of those directed their queries to our DNS server and were thus detected using *PassiveDNS*.

6.1 HONEYPOT

We have a number of aspects that we analyse on in the honeypot, as described in Table 4. To observe the most popular services, the first aspect is whether a malware sample connected to the honeypot. The second is to which service the malware sample tried to connect. The last is whether it tried to upload a file (e.g. a replication of the malware itself) to the honeypot. Systems that make connections, or interact with the honeypot and ultimately systems that transfer files to a honeypot are suspected to be infected with malware. We have had zero connections to the honeypot, in other words: no malware sample attempted to connect to the honeypot. Therefore, the other two aspects also have zero malware samples that correspond to it. A reason for which no connections are made to the honeypot, is that the malware starts to connect to the local network machines after three minutes of execution time, the time that we concluded was enough time for the malware sample to infest the workstation and the network (see Section 5.2). To validate that this is not related to the three minutes execution time, we ran the first 50 malware samples for a

Table 4: Aspects on which the results are analysed.

Category	Aspect	# samples
General	Interacted with network	82
Honeypot	Connected to honeypot	0
	What services are reached by malware	0
	Uploaded file to honeypot	0
DNS data	Issued DNS request	68
	Issued DNS request at our server	67
	Issued DNS request at another server	1
	Domain name is candidate for DGA	5
	Does the domain name request yield a NXDOMAIN	38
Flow data	Only issued DNS request	1
	Connected to IP address without issuing a DNS request	14
	Issued DNS request before connecting	68
	Connected to 1.2.3.4	67
	Connected to other IP address	27
	Connected to non-standard port	28

second time, now with one hour execution time, the execution time that we would try in case the three minutes proved not to be enough. In this second run, there where still no connections to the honeypot. This leads us to the conclusion that today a server honeypot is not an efficient tool to detect malware on a network.

6.2 DNS DATA

The set of domain names in the logs of the tcpdump packet capture is a superset of those contained in the logs of *PassiveDNS*. We have supplemented the *PassiveDNS* logs with the DNS replies from the packet capture that were not directed to our DNS server. As described in Section 5.3, this is the same result as obtained by running two instances of the *PassiveDNS* tool, one close to or on the DNS server, the other close to or on the border gateway and then matching the information of both logs files to each other. Doing this results in a complete overview of all DNS requests that are done in the closed environment.

Of all 82 samples that interacted with the network, 68 queried a DNS server, ours (67 samples) or a remote one (one sample), for resolving a domain name. The domain names that were queried are listed in Table 5. The number of times we have seen the domain names adds up to more than the amount op samples that have accessed the network. This is because a malware sample queries one or more domain for one or more times within its execution time. There were eight domain names that were queried by more than one sample. These are the bold domain names listed in Table 5. Only two of them resolve on January 14, 2014.

As botnet malware is getting more and more common [21], and botnets using more and more DGAs [3], we had expected to see more malware samples that query DGA-generated domain names, but there are only five such candidate domain names in the list, the ones that are unpronounceable. They are listed in Table 5, showed italic. The other domain names suggest their self-describing their goals. There are a lot of domains that end in `no-ip.org`, which is a well-known provider of Dynamic DNS. Dynamic DNS is a service that points a domain name to a dynamic IP address, so this technique can be used for IP-fluxing [52, 79, 69, 26], switching the IP address in an A DNS record of a domain very frequently, in order to evade IP blocking.

We first describe our results in light of the feature classification of Bilge *et al.* (see Table 2), as described in Section 5.3. Their DNS answer-based and TTL value-based are not applicable to our experiment, because in our experiment, the network does not have a connection to the Internet. From our own DNS server, the workstation gets fake DNS answers, so the workstation does not get provided with

Table 5: List of queried domain names and the amount of requests to that domain (over all malware samples). The domain names shown in bold face are queried by more than one malware sample. The domain names shown in italic face are candidates to be generated by DGAs.

Domain name	Amount	Domain name	Amount
adf.ly	2	airforce.dyndns.biz	2
api.wipmania.com	6	childhe.com	6
core.mochibot.com	2	customer.cc.at.paysafecard.com	2
darnnlogs.no.ip.org	14	df5.no-ip.info	14
doser.no-ip.info	16	**downloads.fcuked.me.uk**	16
dveskrepki.ru	2	findcopper.org	2
findwarm.org	2	firstnationarts.com	2
ftp.drivehq.com	4	ftp.tripod.com	4
furzkissen.selfip.com	4	hawet.zapto.org	4
holderman.hopto.org	2	hstnm1.dontexist.net	2
imarcoseduardo.no-ip.org	36	img193.imageshack.us	36
img580.imageshack.us	2	irc.webchat.org	2
kabutokiller.no-ip.info	16	ksamapepito.no-ip.org	16
l3asel.no-ip.org	16	markinyourdark.no-ip.org	16
maxrepjoaki.no-ip.biz	10	mise1.zapto.org	10
monzterddos.no-ip.info	12	movieartsworld.com	12
mqcbpkzjghjt.com	6	***mqcbpkzjghjt.net***	6
please23.zapto.org	14	poni.no-ip.biz	14
promos.fling.com	1	r2crystal.narod.ru	1
ratmehard.no-ip.org	2	relaxedclick.com	2
searchdepressed.org	7	searchelastic.org	7
searchfertile.org	3	securytbr4455.sytes.net	3
smtp.gmail.com	1	sportfishingarts.com	1
sssss.no-ip.biz	24	track.installtrack.info	24
tudoafro.com	4	**ulisessoft.info**	4
update-key.com	4	visualbasic.pro.br	4
wootwootrs.no-ip.org	2	www.aamailsoft.com	2
www.google.at	1	www.mochiads.com	1
x.mochiads.com	2	***xgukreqwpbqte.com***	2
xgukreqwpbqte.net	8	xz69.no-ip.info	8
yah-crackers.no-ip.org	12		

real DNS records. The time-based and domain name-based features are based on the client-side of DNS, as they consist of features like the frequency a client requests that domain name. Time-based features include the frequency of querying a domain, which we cannot base conclusion on, because we only run a sample for three minutes. Nevertheless, there are malware samples that do repeatedly query a domain name. For instance, one malware sample queried `l3asel.no-ip.org` eighteen times in three minutes (whilst only trying to make a connection to the remote system only nine times) and another queried `xz69.no-ip.info` 24 times whilst only connecting to the server six times. It could be that the malware expects a certain IP address when resolving a domain name, and therefore keeps trying. The domain name-based features include the ratio of numerical characters and the ratio of the length of the Longest Meaningful Substring (LMS). The numerical character method is used for domains that look like being generated by a DGA. As this method looks for the ratio of numerical characters to alphabetical characters, this method will not yield us DGA-generated domain names, as the domain names in our dataset do not have large differences in this ratio. The LMS method yields results. This method is based on the meaning of DNS: providing human-readable names for IP addresses. This means that the website of a company will most likely have the name of the company in the domain name. To have an example, it is likely that the Bank of Ireland uses the domain name `bankofireland.com`. Using Google to match a domain name with the title of the website can be useful for looking whether a domain name that is frequently requested, should be requested that often [7]. In our data set, almost all domain names do not have a long LMS in it, so automated detection would more likely mark the domain names to be involved with malware.

Applying the *NXDOMAIN method* from literature [3, 27, 76] , did not yield reliable results. 38 of the total 62 domain names did not resolve to an IP address in our experiments. A possibility is that the services, that were once located at one of the not resolving domains, are now moved to another domain, or taken down. Either way, applying the *NXDOMAIN method* in retrospect does not have to yield the same result as when the malware was active on the Internet. The DNS Census 2013 dataset contains DNS records that were registered in the past, which enables one to apply the *NXDOMAIN method* in retrospect [17]. We cannot conclude why domain names do not resolve at this time. Which domain names did and did not resolve is stated in Table 6. By applying the *NXDOMAIN method* in retrospect, we cannot base conclusions on this, as domains that did resolve at the time that the malware was in the wild, may not be reached at this time.

Table 6: *NXDOMAIN method* results, executed on 2013-12-11.

NXDOMAIN	Resolving
airforce.dyndns.biz	adf.ly
darnnlogs.no.ip.org	api.wipmania.com
df5.no-ip.info	childhe.com
downloads.fcuked.me.uk	core.mochibot.com
findwarm.org	customer.cc.at.paysafecard.com
firstnationarts.com	doser.no-ip.info
furzkissen.selfip.com	dveskrepki.ru
hawet.zapto.org	findcopper.org
holderman.hopto.org	ftp.drivehq.com
hstnm1.dontexist.net	ftp.tripod.com
imarcoseduardo.no-ip.org	img193.imageshack.us
imarcoseduardo.no-ip.org	img580.imageshack.us
kabutokiller.no-ip.info	irc.webchat.org
ksamapepito.no-ip.org	poni.no-ip.biz
ksamapepito.no-ip.org	promos.fling.com
l3asel.no-ip.org	r2crystal.narod.ru
maxrepjoaki.no-ip.biz	relaxedclick.com
mise1.zapto.org	gmail-smtp-msa.l.google.com
monzterddos.no-ip.info	sssss.no-ip.biz
movieartsworld.com	ulisessoft.info
mqcbpkzjghjt.com	www.aamailsoft.com
mqcbpkzjghjt.net	www.google.at
please23.zapto.org	a90.g.akamai.net
please23.zapto.org	x.mochiads.com
ratmehard.no-ip.org	
searchdepressed.org	
searchelastic.org	
searchfertile.org	
securytbr4455.sytes.net	
sportfishingarts.com	
track.installtrack.info	
tudoafro.com	
update-key.com	
visualbasic.pro.br	
wootwootrs.no-ip.org	
xgukreqwpbqte.net	
xz69.no-ip.info	
yah-crackers.no-ip.org	

The obtained flow data contains all network traffic that traversed the bridge from the start of each experiment until the end. This shows that there were 82 samples that interacted with the network. As said in the previous section, many of these make use of DNS, and could be identified by that detection approach. As our closed environment is not connected to the Internet, we cannot apply the flow size-based features proposed by Bilge *et al.* [7]. The fact that we only run the malware samples for three minutes, restricts our use of the temporal features. However, we can apply the client access pattern-based features, by looking at IP addresses and port numbers to which the malware samples connect.

In the flow data, there are 14 samples that did not make use of DNS, but did interact with the network. These samples have a preconfigured IP address in their source code. This means that the malware does not use fluxing, and can therefore be easily blocked by blocking the IP address. The other 68 samples first issued a DNS request. 67 of these connected to our forged 1.2.3.4 IP address thereafter. 27 malware samples connected to another IP address (partly the same samples). These samples make use of domain names, so can be using fluxing.

The flow data gives us another piece of information that the other detection approaches do not, namely the port numbers. After having received an IP address from a DNS server, the malware will start to connect to that IP address. The port numbers are very different, although the transport protocol is always TCP. Some malware uses port 80, the HTTP port, but port 60, 8080, 81, 3174, and 1604 are also present in our data set. The port numbers we have seen connections to on our forged IP address 1.2.3.4, and their assigned uses [30, 31] are in Table 7. As we know for sure we only deal with malware, we can safely say that the ports are not used for their assigned purpose. We can hold this list next to the most used ports list of nmap[1], a famous port scanner. It lists port 80, 23, 443, 21, 22, 25, 3389, 110, 445, and 139 as the top 10 used TCP ports. We define ports not on this list as non-standard ports, of which it is unlikely that normal software would use these ports. The list shows that most of the port numbers in our dataset are non-standard, which means that they are suspect to be used for malicious activities. 28 malware samples connected to one or more non-standard ports.

We have also seen IRC botnet malware. These samples first query the IRC server `irc.webchat.org`, and after getting the IP address, connect to that IP address on port 6667 (the assigned port for IRC). This is the traditional example of C&C malware [39], and is therefore suspicious.

1 http://nmap.org/

Table 7: Port numbers of connections to 1.2.3.4 and their assigned uses.

Port	Assigned use
0	Reserved
21	FTP
80	HTTP
81	Unassigned
91	MIT Dover Spoiler
200	IBM System Resource Controller
443	HTTPS
465	URL Rendesvous Directory for SSM
888	AccessBuilder
999	Unassigned
1337	menandmice DNS
1604	icabrowser
2000	Cisco SCCp
3085	PCIHReq
3086	JDL-DBKitchen
3170	SERVERVIEW-ASN
3174	ARMI Server
3175	T1_E1_Over_IP
4662	OrbitNet Message Service
5312	Permabit Client-Server
5315	HA Cluster UDP Polling
5317	HP Device Monitor Service
6667	IRC
6697	Unassigned
25567	Unassigned

6.4 CORRELATING THE RESULTS

In this subsection, we will assess the synergy of combining the detection approaches, as is the goal of this work. The combined approach can be used next to the detection approaches on their own, like the *NXDOMAIN method* from the DNS data and the characteristics from the flow data.

As the honeypot did not receive any connections from the malware samples, but the DNS server and the flow data exporter did, we can hypothesize that the focus of malware today is more on C&C or phone-home technology. Domain names ending in `no-ip.org` (see Table 5) are example suspects of C&C servers. When the honeypot would have received connections, that information could also be correlated to the DNS and flow data like in Section 6.4.

DNS and flow data can be combined to give a better impression of infected systems in a network. As we have seen in Listing 1, the *PassiveDNS* log shows the domain name that is queried and the IP address it results in. These IP addresses can be matched to those in the flow data. When traffic is seen to non-standard ports (like in Listing 2, with port 1337) or traffic that is characteristic for botnet malware, there is an additional reason to qualify the system in the network the traffic originates from is infected with malware. An example from our dataset first issued a DNS query to `furzkissen.selfip.com`, to which our DNS server responds with 1.2.3.4. The malware subsequently connects to 1.2.3.4:1337. Our *PassiveDNS* reports that `selfip.com` is used, another known Dynamic DNS provider, while the flow data sees traffic to 1.2.3.4 on port 1337. Our combined approach can link the domain name information and the flow data and conclude that there are multiple reasons for marking the workstation as infected, and therefore isolate the workstation from the network, ensuring it cannot connect to the Internet any longer, and rendering it unable to infect other machines.

6.5 SAMPLES THAT STOOD OUT

Some samples generated some results that were unlike the other results. In this subsection, we will describe what made these samples stand out, and give an explanation.

In the whole data set, there were only seven samples that are candidate to use a DGA. Because of all the recent research into malware that uses DGAs and the fact that these botnets were recently discovered and taken down (e.g. Conficker), we had expected to see more of these generated domain names in our DNS data set.

One malware sample connected to `crl.microsoft.com` and `crl.verisign.com`. These are the Certificate Revocation List servers of Microsoft and Verisign, which are used as one of many methods to check whether SSL certificates are no longer valid. An explanation that these servers appear, is that the malware uses the SSL library of Windows XP, and that Windows XP, on its part, checks these lists.

There was one malware sample that tried to connect to `smtp.gmail.com`, on port 465, the port that Gmail uses for SMTP over SSL. This means that the malware is probably trying to send an email. Google requires users of the SMTP server to login, so it also means that the login credentials for the SMTP server must be included in the malware, or that sending an email will always fail.

As seen in Table 4, there was one malware sample that only issued a DNS request. It is the same malware sample as the one that did not use our DNS server for resolving a domain name. As the external DNS servers could not be reached, no DNS answer was received by the sample, making it impossible to know to which IP address it should connect.

7 CONCLUSIONS

In the past pages, several malware detection approaches are discussed, and combining them in order to achieve synergy in detecting malware is investigated. We looked at the state of the art of detecting malware infected systems by using honeypots, DNS data and flow data and conducted an experiment in which we mimicked a corporate network with a workstation that got infected with malware. Our honeypot did not receive any connections, but the DNS data and the flow data can be combined to base the decision whether a system is infected or not on the results of multiple approaches. From our results, we conclude that combining multiple malware detection approaches can give information for a better informed decision whether a workstation is infected with malware or not, by marking it as infected by more than one approach, and correlating these sources of information.

In Chapter 2, Chapter 3, and Chapter 4, we investigated the literature on detecting malware with honeypots, DNS data and flow data. We concluded that for honeypots, there are numerous kinds of honeypots, and that we needed a server honeypot, that is medium-interaction. When running our experiment, we have had zero connections to our honeypot, and we conclude that a honeypot is not an effective tool for malware detection. For the DNS data in a closed environment, applying the *NXDOMAIN method* was not applicable to our dataset. We have looked at domain names suspected of being generated by DGAs and statistical properties of the domain names, and have concluded that DNS data analysis is a helpful tool for malware detection. Being in a closed environment cannot yield the same traffic characteristics as being on the Internet, as the malware samples cannot reach the servers they can reach on the Internet. Therefore, we looked at port numbers to which the malware connects, and whether the IP addresses to which the samples connect were hardcoded in the malware or requested via DNS. We have seen that 28 of 82 malware connects to non-standard ports.

Combining flow data analysis and DNS data analysis achieves a better informed decision whether a system in the network is infected by malware. Systems that issue a DNS request for a suspicious domain name, and moments later try to connect to the associated IP address on a port that is non-standard, provide the our combined approach with multiple reasons to conclude that the system is infected by malware.

This research can be carried on by combining the same approaches on a real network, not using a set of malware samples, but normal traffic, in which malware is included. This way, the *NXDOMAIN method* can be used directly on the data, which gives accurate results. In the flow data, more characteristics, such as described in Gu *et al.* [25] than just the used port numbers can be found, because the malware will connect to the right servers, instead of our forged IP address 1.2.3.4.

Another idea for future work consists of choosing other detection approaches, such as SNMP data analysis, to detect malware. In this research, we have chosen three approaches that are already widely used in corporate networks. Combining other approaches may prove to be very efficient in detecting malware correctly.

BIBLIOGRAPHY

[1] Y. Adachi and Y. Oyama. Malware Analysis System using Process-level Virtualization. In *2009 IEEE Symposium on Computers and Communications*, number Vmm, pages 550–556. IEEE, July 2009.

[2] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *USENIX Security Symposium*, 2011.

[3] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[4] R. Baumann and C. Plattner. Honeypots. Technical report, Swiss Federal Institute of Technology, 2002.

[5] R. Berthier, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, D. Sheleheda, P. Ave, and F. Park. Nfsight: NetFlow-based Network Awareness Tool. In *Proceedings of the 24th USENIX LISA*, 2010.

[6] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. Disclosure: Detecting Botnet Command and Control Servers Through Large-Scale NetFlow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[7] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *NDSS*, 2011.

[8] M. Brunner, C. M. Fuchs, and S. Todt. AWESOME - Automated Web Emulation for Secure Operation of a Malware-Analysis Environment. In *SECURWARE 2012 , The Sixth International Conference on Emerging Security Information, Systems and Technologies*, number c, pages 68–71, 2012.

[9] D. Bruschi, L. Martignoni, and M. Monga. Detecting Self-mutating Malware Using Control-Flow Graph Matching. In *Detection of Intrusions and Malware \& Vulnerability Assessment*, pages 129—-143. Springer, 2006.

[10] S. Campbell, S. Chan, and J. R. Lee. Detection of Fast Flux Service Networks. In *Proceedings of the Ninth Australasian Information Security Conference - Volume 116*, pages 57–66, 2011.

[11] C.-m. Chen, S.-t. Cheng, and R.-y. Zeng. A Proactive Approach to Intrusion Detection and Malware Collection. *Security and Communication Networks*, 6(7):844–853, July 2013.

[12] D. Chiba, K. Tobe, T. Mori, and S. Goto. Detecting Malicious Websites by Learning IP Address Features. In *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, pages 29–39. IEEE, July 2012.

[13] D. Chiba, K. Tobe, T. Mori, and S. Goto. Analyzing Spatial Structure of IP Addresses for Detecting Malicious Websites. *Journal of Information Processing*, 21(3):539–550, 2013.

[14] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, pages 715–720. IEEE, Oct. 2007.

[15] R. Curtmola, A. Del Sorbo, and G. Ateniese. On the Performance and Analysis of DNS Security Extensions. In *4th International Conference, CANS 2005, Xiamen, China, December 14-16, 2005. Proceedings*, pages 288–303. Springer Berlin Heidelberg, 2005.

[16] L. Deri. nProbe: an Open Source NetFlow Probe for Gigabit Networks. In *TERENA Networking Conference (TNC 2003), Zagreb, Croatia*, 2003.

[17] DNS Census. DNS Census 2013, 2013.

[18] M. Drozd, M. Barabas, M. Gregr, and P. Chmelar. Buffer Overflow Attacks Data Acquisition. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, pages 775–779. IEEE, Sept. 2011.

[19] European Network and Information Security Agency. Proactive Detection of Security Incidents II - Honeypots, 2012.

[20] M. Feily, A. Shahrestani, and S. Ramadass. A Survey of Botnet and Botnet Detection. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 268–273. IEEE, 2009.

[21] J. François, S. Wang, R. State, and T. Engel. BotTrack: Tracking Botnets Using NetFlow and PageRank. pages 1–14, 2011.

[22] J. Giles. Conficker: the Enemy Within. *New Scientist*, 202(2712):36–39, 2009.

[23] J. Göbel. Amun: Automatic Capturing of Malicious Software, 2010.

[24] A. R. A. Grégio, I. L. Oliveira, R. D. C. Santos, A. M. Cansian, and P. L. de Geus. Malware Distributed Collection and Pre-classification System using Honeypot Technology. In B. V. Dasarathy, editor, *Proc. SPIE volume 7344*, volume 7344, pages 73440B–73440B–8, Apr. 2009.

[25] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, pages 139–154, 2008.

[26] F. Haddadi and A. N. Zincir-Heywood. Analyzing String Format-Based Classifiers For Botnet Detection: GP and SVM. In *2013 IEEE Congress on Evolutionary Computation*, pages 2626–2633. IEEE, June 2013.

[27] S. Hao, N. Feamster, and R. Pandrangi. Monitoring the Initial DNS Behavior of Malicious Domains. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference - IMC '11*, pages 269–278, New York, New York, USA, 2011. ACM Press.

[28] A. Hassan and M. A. Ali. Collecting Malware from Distributed Honeypots — Honeypharm. In *2011 IEEE GCC Conference and Exhibition (GCC)*, pages 351–352. IEEE, Feb. 2011.

[29] Honeynet Project. Server Honeypots vs. Client Honeypots. http://www.honeynet.org/node/158, 2008. Accessed on 2013-12-04.

[30] IANA. IANA Service Names and Port Numbers. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt, 2013. Accessed on 2013-12-11.

[31] Internet Engineering Task Force. RFC6335: Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. http://www.ietf.org/rfc/rfc6335.txt, 2011. Accessed on 2013-12-11.

[32] Internet Engineering Task Force. RFC7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. http://tools.ietf.org/html/rfc7011.html, 2013. Accessed on 2014-01-07.

[33] J. Kohlrausch. Experiences with the NoAH Honeynet Testbed to Detect new Internet Worms. In *2009 Fifth International Conference on IT Security Incident Management and IT Forensics*, pages 13–26. IEEE, 2009.

[34] T. Krueger, N. Krämer, and K. Rieck. ASAP: Automatic Semantics-Aware Analysis of Network Payloads. In *Privacy and Security Issues in Data Mining and Machine Learning*, pages 50–63. 2011.

[35] S. Kullback. The Kullback-Leibler distance. *The American Statistican*, 41(4):340–341, 1987.

[36] S. Kumar, R. Sehgal, and J. S. Bhatia. Hybrid Honeypot Framework for Malware Collection and Analysis. *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, pages 1–5, Aug. 2012.

[37] E. Lacombe, F. Raynal, and V. Nicomette. Rootkit modeling and experiments under Linux. *Journal in Computer Virology*, 4(2):137–157, Oct. 2007.

[38] V. Levenshtein. Efficient reconstruction of sequences. *IEEE Transactions on Information Theory*, 47(1):2–22, 2001.

[39] H.-C. Lin, C.-M. Chen, and J.-Y. Tzeng. Flow Based Botnet Detection. In *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pages 1538–1541. IEEE, Dec. 2009.

[40] P. R. Marupally and V. Paruchuri. Comparative Analysis and Evaluation of Botnet Command and Control Models. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 82–89. IEEE, 2010.

[41] B. McCarty. The Honeynet Arms Race. *IEEE Security & Privacy Magazine*, 1(6):79–82, Nov. 2003.

[42] I. Mokube and M. Adams. Honeypots: Concepts, Approaches, and Challenges. In *Proceedings of the 45th annual southeast regional conference on - ACM-SE 45*, pages 321—-326, New York, New York, USA, 2007. ACM Press.

[43] A. Moser, C. Kruegel, and E. Kirda. Exploring Multiple Execution Paths for Malware Analysis. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 231–245. IEEE, May 2007.

[44] C. Musca, E. Mirica, and R. Deaconescu. Detecting and Analyzing Zero-Day Attacks Using Honeypots. In *2013 19th International Conference on Control Systems and Computer Science*, pages 543–548. IEEE, May 2013.

[45] Net Applications.com. Desktop Operating System Market Share. http://netmarketshare.com/operating-system-market-share.aspx, 2013. Accessed on 2013-11-25.

[46] Network of Affined Honeypots Project. European Network of Affined Honeypots. http://www.fp6-noah.org/, 2005. Accessed on 2013-12-15.

[47] Network Working Group. RFC3917: Requirements for IP Flow Information Export (IPFIX). http://www.ietf.org/rfc/rfc3917.txt, 2004. Accessed on 2013-12-09.

[48] Network Working Group. RFC3954: Cisco Systems NetFlow Services Export Version 9. http://tools.ietf.org/html/rfc3954.html, 2004. Accessed on 27-10-2013.

[49] S. Newman. How & Why DGA's Evade Your Corporate Security Controls. http://www.prodevmedia.com/FSISAC/2012/fall/21_StephenNewman_Stopping_the_New_Wave.pdf, 2012. Accessed on 2013-07-01.

[50] Offensive Security Ltd. Metasploitable. http://www.offensive-security.com/metasploit-unleashed/Metasploitable, 2013. Accessed on 30-10-2013.

[51] A. Pauna. Improved Self Adaptive Honeypots Capable of Detecting Rootkit Malware. In *2012 9th International Conference on Communications (COMM)*, pages 281–284. IEEE, June 2012.

[52] P. Porras. Inside Risks: Reflections on Conficker. *Communications of the ACM*, 52(10):23, Oct. 2009.

[53] P. Porras, H. Saïdi, and V. Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[54] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 15—-27, New York, New York, USA, 2006. ACM Press.

[55] A. Ramachandran, N. Feamster, and D. Dagon. Revealing Botnet Membership Using DNSBL Counter-Intelligence. In *Proc. 2nd USENIX Steps to Reducing Unwanted Traffic on the Internet*, pages 49–54, 2006.

[56] K. Ramachandran and B. Sikdar. Modeling Malware Propagation in Networks of Smart Cell Phones with Spatial Dynamics. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 2516–2520. IEEE, 2007.

[57] R. Real and J. M. Vargas. The Probabilistic Basis of Jaccard's Index of Similarity. *Systematic Biology*, 45(3):380, Sept. 1996.

[58] P. Royal. Analysis of the Kraken Botnet, 2008.

[59] A. Schonewille and D.-J. van Helmond. *The Domain Name Service as an IDS*. PhD thesis, 2006.

[60] V. Sharma. An Analytical Survey of Recent Worm Attacks. *International Journal of Computer Science and Network Security*, 11(11):99–103, 2011.

[61] S. Shevchenko. Srizbi's Domain Calculator. http://blog.threatexpert.com/2008/11/srizbis-domain-calculator.html, 2008. Accessed on 29-10-2013.

[62] S. Shevchenko. Domain name generator for murofet. http://blog.threatexpert.com/2010/10/domain-name-generator-for-murofet.html, 2010. Accessed on 29-10-2013.

[63] M. Skrzewski. Monitoring Malware Activity on the LAN Network. In *Computer Networks*, pages 253–262. 2010.

[64] M. Skrzewski. Flow Based Algorithm for Malware Traffic Detection. In *Computer Networks*, pages 271–280. 2011.

[65] M. Skrzewski. Network Malware Activity – A View from Honeypot Systems. In *Computer Networks*, pages 198–206. 2012.

[66] L. Spitzner. Dynamic Honeypots. http://www.symantec.com/connect/articles/dynamic-honeypots, 2003. Accessed on 2013-10-24.

[67] L. Spitzner. Honeypots: Definitions and Value of Honeypots. http://www.tracking-hackers.com/papers/honeypots.html, 2003. Accessed on 2013-10-24.

[68] D. Srinivasan and X. Jiang. Time-Traveling Forensic Analysis of VM-Based High-Interaction Honeypots. In *Security and Privacy in Communication Networks*, pages 209–226. 2012.

[69] E. Stalmans and B. Irwin. A Framework for DNS Based Detection and Mitigation of Malware Infections on a Network. In *2011 Information Security for South Africa*, pages 1–8. IEEE, Aug. 2011.

[70] J. Steinberger, L. Schehlmann, S. Abt, and H. Baier. Anomaly Detection and Mitigation at Internet Scale: A Survey. *7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2013, Barcelona, Spain, June 25-28, 2013. Proceedings*, pages 49–60, 2013.

[71] J. Stewart. Bobax trojan analysis. *SecureWorks*, 17(May), 2004.

[72] B. Stone-gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet : Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647, 2009.

[73] M. Szczepanik and I. Jóźwiak. Detecting Malwares in Honeynet Using a Multi-agent System. In *Networked Digital Technologies*, pages 396–401. 2010.

[74] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel. BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT '12*, pages 349–360, New York, New York, USA, 2012. ACM Press.

[75] P. Čeleda, J. Vykopal, T. Plesník, and J. Novotný. Malware Detection From The Network Perspective Using NetFlow Data. In *3rd NMRG workshop on NetFlow/IPFIX usage in network management*, 2010.

[76] 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*, number 1, pages 476–481, 2008.

[77] B. Weymes. DNS anomaly detection: Defend against sophisticated malware. http://www.net-security.org/article.php?id=1844, 2013. Accessed on 30-10-2013.

[78] G. Wicherski. Placing a low-interaction honeypot in-the-wild: A review of mwcollectd. *Network Security*, 2010(3):7–8, Mar. 2010.

[79] S. Yadav, A. K. K. Reddy, a. N. Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, pages 48–61, 2010.

[80] T.-F. Yen and M. K. Reiter. Are Your Hosts Trading or Plotting? Telling P2P File-Sharing and Bots Apart. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 241–252. IEEE, 2010.

[81] M. N. Yusuff. Honeypots Revealed, 2005.

A LIST OF MALWARE SAMPLES

Table 8: List of the 997 malware samples executed on the workstation.

MD5 hashes	
002485852df093134f18288492ea1a59	0032469d1f8ca921f1f14ec04ef0b209
003e845bdcc5367220bf13f7170da16f	00c28cee9c6874302982045b5faff846
00c9833a35b0a8bd957dd85c41fce5b9	00e6af8a522259b83fa317b7a2f8f161
010025ab068e4744a644d1ad29e981a1	014a9cb92514e27c0107614df764bc06
01652609d8c786fe8b39c7aded8b8fdf	017bea72340e230c96dbbcac36031fa2
01aa11d5a865a9c34270b56e5a542a86	01ae30f6635fc661a7f3a4995962e83f
01db10a317194fe7c94a58fae14f787c	02b94f96e0f1bc9189b756edd8b64318
02cf380fa8ff92dc6e74eaf188575f3e	02fb8545f6940114e4d5c10ed777a04c
03201a9f9c06e42159d98ccf2719d8af	035c26c52bb1ead2ddea66b9f0ec9261
0380643698ce56a7f614021b0856c5d5	03831a13d480daed3d2b63201cb6bcc5
03e5b8d5b2696cf34359e1b8d2243da0	04085a9bdd41dd53c40056f2ebb0ea8e
0410d705efb224a007bd5b675ef42169	0455dafdfd07a9849b143209db253b56
04b3194bce294556586c87b627ccdae3	04c5850be571a95b1ca5563e7498c7be
04cf0389139862f627dfa7ab643d2655	04d2ddc47267352028cad0070d4e9ac4
04d56751f25d6169005395ccd13eae55	04d6066541c0292dc6e9897b4f85593f
0546abe6293ba40348e1734fafca47ec	055fa0bb647b4f2277c9236e2310795c
0579133b12b454ab568d60609f041d32	05887d3c42e7f186f2a59fce85af6e75
05b6f4b41231d437216c87f7c752dc8d	05cf6e4558cf04c2d546df2da9d57e76
0605cb4765c0036e7f6b46d016a1bd1c	063996fd1db100ea7501d7af9847e1ba
06453466ed8f14941cd211388ee00fa2	064ae6c4099945f77249566a57c8564a
067bed758bf971259c2c039ae8ed7193	06988c2a3f0bd9a8b10279191fb9039c
06b125b04dd69a5aabf76e2cf48f4b0f	06e54162b8b0324232fbf820c0c22496
0709fd721e486fc3091542ff7e4a0b49	070d303c95856722bea316d01b42df46
07df3e2401936a063ab27fcdb32dac99	07f398af7e2b789d550b8b3e6f0465e4
0809eb81c3d061d637df4ca6f3ae62fb	08152df537bceb359d9ef3178e29fa5a
0826f81f22867a464021aa2f94576693	082afd83fc8bfe64479651a7bc924a20
084fca631cba38858ae3a00cf0001882	085412f65a35d71d8ffba5bec6632997
0854d8197846b12e6ea64e83007f50d9	088565f653b5d95e33adfd4833e12ee6
08b7da45b3a9dd5d4108f42708e64203	08daa9d5d81e6f12f8995f5162d7e415
09511410b0c4333c2399703c56e36868	095df20ced77e89ab1ff0c01b490f622
0979d6e554e66a421ee7b6675ab409a9	097c051268fc5929773a9c3cff788c1c
09be8c337ac66ab525bd5f715547f9fa	09d49c997fa5df14cbefd9b745e04acf
09fdf5ee81408995fe6538dc826b554a	0a13612f52c6ceb541c0144d3cdb1947
0a83777e95be86c5701aaba0d9531015	0ac8ae925d4e7d6b754c6533068b6e06
0ac9b76b75dc91d42e0ca83a4890cbe0	0b147d5f4fcf4c665f142d6209fa1cb2
0b346b201da259377ba11438504bbd9e	0b3dbac1f5461615b288e1c9076e7176
0b644ebe34259c653f7ca3c340af4da9	0b978cd6214cdf20e1ae2cce0ba173df
0bb0eb44a8d7505e44b03633fe9c4259	0bdfd0a6b3533222ae9e86fb5f19f111
0be7c6f82bc7b3cfecd8a158ca2ef9ad	0c16055804fcb734a0bcb0bf9bac98b5
0c440c4536f18ef5258b5ab4c65d02e8	0c558b3f646c7e8cb1ac0a17ac9db924

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
0cec129e2693cea14b83b121a7250f1c	0d6ce9c25d44e198e3f96c846c472227
0d9c6f4beaf23a35b74c1c9f0927105e	0dd7e0e00ecb6b23449e720d2f6fcb5e
0e15851d24b2b4d208c96523ff98b6c0	0e7be3bb556309a1310bedce7924bdc1
0ec2a2f82efdb8f163f3fde5ac0012f6	0f0554e3a8dc4938a2dff13b6cb080a9
0f068e82a6e139884f691c4211292213	0f104f52e701132268f2ebfdd0d3de25
0f1b37d10c2b7a148b4fa329041e0675	0f4813e358687324b9de239011d822a6
0f4ae8eaa21463cc59abc83611517eaf	0f5993c32b042816e1f3deb08b6e2af3
0f6441892cfd2ab54bd5dae53e92239e	0f9561f26e4ee69b31f6bdb57f9e6c8d
0fef3c0c12f89c20ea09d99f5f6ead0a	0ffae323037b4845e1c2dd8a11e0af01
101a14dbfd11b04551dc661228b99e3b	1060d4dde3d693284d4665696776a54d
107ac0e409ded4e6bc358167f2b76e75	109bf42d3a6c45ed704bf0c62bcf7b89
10f60a69da1dcd91b69d71b831d41feb	114d598405fc691ce652323d14e496c7
11765af7e1f7c0ddeeaaacd245c6d42d	118f67cfdbed578a729ef633cc19325a
119c71e12bc9bf6b0f26d09c82d38d60	121201bd8ec4b48031566b8195694954
12224937103c1d0fa3efd808bc491c4e	123675f34b782e33d45a3a00f9c3a350
123c27059bb48cf72f253da31f67575f	126e17d9b096b8ecf9afcb89a43d35b7
128879f57b59cf7f1db2038f539db945	12cecc3c14160f32b21279c1a36b8338
12d2388a8ce019ca6792b9a87deea374	130b6895069434a86dfa55d2f785b4d1
13196e1139e2dbadd3db97d335d517b7	1379b9ce1475834e672b90fc2611ef6f
137d7b025ad99a60024dfed7e8b8b101	13f90b323b0acdece64fe7f41126e675
1426fe0324624ace16d4b769778f0bf2	14546231ec7e0c7dea099633d95f0072
148891362ae542a3163d37bcefacee0de	14c2946528e8ea084c39de37afb14b98
14cd24f0f7c32f942c8e16f5eac2bbba	15075aba80fc63b613e07ddf9cc2159c
154ad2f8b8bf6df37e684e14b0ffe3f6	154ce466aedbe66ef86f6bc60b05c8be
1574ee69c6445bf8e0fd26f2e9a703c0	16237f5b7e7d7bd7b2d1d26d37073a6c
16495486e03ae792a2221ebea069dff5	1666316c1db9e6051990574e3e25edee
1690cab8d0502df13ac27e15867c806f	16a6955696ef375f1efb1d371cd9928c
16f5796cd816eb27dbb5786fd63b00b7	16fa006e5b5d50cf16dc8ded41ecafe1
17b8809764f6c1122c19e730f9ad4540	17cd16dcbb45bf64317e638ac4b4c675
186920c608674943292030114d7e3ae8	18764f36686a678263c72d2451f636ef
18a61b6cf8a8feae6480de1def07e282	199a96e64e86492bb1c3a617decd66d8
19a90a060e71b81c0b2cd21b1d092e38	19b911c5e0f4c36ac3d511306b29878e
19d279af28135c865fab6103b9b32cc7	19d5ad2adb3038238f877e430bc7727d
1a5d3021db16ee2670c8e3d3b9a0771f	1a6031959d62cc35e0c9d95a8eadfa38
1a7ab67a80403956bf10ca84a8410cb6	1a985147e4ab082a3f4da52a27525aa4
1ab5e53c6524fddc8b793bcea1f47d5f	1abae8bece6b2aa93173df4b08b65ada
1ae1f009df1679433057d09b7a4e1c4f	1b2dc2331e6dce4f8a6edc0a335b3a55
1b5b95995db845cc1e4b1b5d9d952198	1b7c0b64ba5d97af1374f6a7e353867e
1bced6319358f60260ad018823113e94	1c0e7503ac0c45086c98302d10ad9887
1c8195a15d83a969e1c7615c108d3035	1cb102e7fd171ec78ab0b2224f102225
1ce5e870656a9cd7b9fcd6d7f63a7771	1d45c718da2b2f7c3c9774eaf5a624b2
1dbd5aec1486919d7a38a3803d1fbcc5	1e134bbfe3a537424a2ccd13d4ed7ea9
1e34b50b8af8dbeb750c291981428053	1e56bc24d8fe21348b29709029add906
1e86e2fdff9c1089b2883fc3ed12b212	1e872bdb352dc6574fc484c7011d892b
1e9509e8363b719d482581cd6ee23f32	1eba72886b84f84ef9d95e93c95465d4
1ecdc7455e30797fbe62a591cd119cd2	1ee8fe3d329999bb7cc1d019eacce75a
1ef1463cb5cd677b3b73e414dc2998af	1f04389bfb1dfaf813a2f67d5cad092b
1f69d3bc800f9f00968b42b49f4ea03b	202c58d508248629244df57603291772

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
2036b2d410630fa85b5a5e53f93289b0	203dd1e8960afd68bde851b682e88c6a
204d8ccdfb33a315760a554a978317e3	20d47409829148715bcce170357810cf
217687daa0a13d4ad824b26225e66c79	22202d6c56cb7ea09ae1ecdce08c71a4
22b5f47308af3841e772d20ef75a235a	22cf18696872ba3975978f4484b89654
231a9252e539ac732d2bd4bac0627ed5	23538b9ee5114655f05b0ce50e3e90d4
23b5e1ddd14b03452af26019d90447b0	23bfb72beae785c3ed75c8f8287a19e3
240aaac273014815852c451d7c016ba9	2465ac937f8d6fc512af438665ecb5c4
24c639a18297db41949983f59e4054fa	24f7bc6eef81a32f620392f8de13ca65
250ea87b50b7cfa2d499b4ab5e2651cc	250fa29286a4022b17ec045c6fe0c1f1
25d4e9e52fe16e5f44237a711d8c6acf	26018892aec8e40915ef9a65265810ee
26083b3d94cb4abe0b1aa443494c0466	260a7503f7d26636ee08fe2ce1264c47
26392892f86b4de826e56eaf8a59d574	265494ea5c259da01a4d77fa84f15d36
26654751cb2e98f5f67aff78d19800eb	267e02cf17f1d34f473a1849fe86011b
26d848591f92601ea328ec86e216aded	26e9b7b495e302fbc6cd160be32602b5
2780d6b4f3f4c4034770a96dea69fc1c	27854044479b7af9acee7d5442b6d720
27b831e93697d45d978796198b421033	27c17e3b1111fc5c3d4f6d779b15d4da
2826ec1db9dd35d4dcfe718a8a4f2115	2867c6976892dc4ec71bcdc6ad933dbc
28b0d90d37faeaa92f937447a82f6313	28c5ada9bdc5ddbbe78b6227c63f0e05
28daf983522cf87af524d22932152d00	28e44bf092f8fa193992fa50aee92c4a
28e8434b81565a4aea9438caf465553a	28f170f63555bbacc1bb30ae058f165b
293906befa51c3ab72a9a94001a69dcb	295c509f093ce76792ab687c0cfb5fa4
29748f96daca16266b9ded60531f916e	299b40e3abdd646f64888cbc450fc637
29b1f56767b437247fe8b1bb81d7de16	29b424c380f8a9015e8877a6aca1d6e6
2a65046feb272b7b5bd825a0a0e57c53	2ac98a19bcb4d9e1061b6e683f59c490
2ae55514d4fd2e80146cdbb1e6941e60	2af3aa23286a216c8995c178490593fc
2b0ca356af63176d70cb976cce626021	2b1bcb854057da7c3b61315af1d65920
2b488c84030f7fe71563793aba9db07f	2bac879b6cdf577969d2f51d4e435289
2bd25e9cf861f0a65e5fbbf0e0b7ab29	2c01a9b096bb1ad949d309c921a8b811
2c174fe0b07600deb5e6c9cf45bd1fa0	2c503bebe612e4f0a590dcde41eafa21
2c75605732e53ab5eda618bcc2a1042d	2c965739bd89c08ea54d886e5babbf7a
2ca1ccd1389576d7a8b6c912320d5faa	2cbffba65a4eb05e79db51579306e988
2cc557e2c58932f2e0c18b683770ba8b	2cca7f7cdcd80e4db2ff923d80b79ac9
2cfa0b1c7ff49727a28c222fd524fc2c	2d3a56e6d3e9e1db867ec24ebb5b9502
2d42417903d1f674b40f7b758da59741	2d6bb17d536a89c08a93ed8055b5d419
2d7839f3fc66dceee05dd4da03474675	2da18dbc1a09de089f53c00cb55544e2
2dcefe60b902897bf4279c041a065bcc	2e0e5bad6caee56856970420c0a996d9
2e4982325d6d34d52ebafa0dd495b466	2ea216e792ef906fef47402a72222763
2ebe59105a5a955361ab3dd16158746d	2f34659e48f529ea4883cfe11c326078
2f3ea5dbb6e57ccdc0d74248f8e9486b	2fceaab4d2f77e4e321130d7c13764e1
301fffbab8f7a6df2da5c892a0e390f7	30428ba182b35376a1564544ab2c562b
30628e8ba8cb728856db1c0a728c6005	3087a840a97c80231d78767ae09bde10
308c2897607d78d6f493f913b627235c	310b183e5cd2318c55a1538741847c1c
311c1c570a45f64739d75cd7c084b50d	3134f4bda9f2bf514b180921b6f3eb0c
31379be1e4c68d5ca66d03c859e7af5f	313f8429df0f599c94ba9fad28b0c5d0
3145dd202cc76c7c3d43b6e572c7643e	314b8a36f24141924d1faa6938088db6
318092c434ce7b8e815085e11955cf8d	31adfb8f43f1598e19249bed709826b3
32881867f46497b5a8db4081d4e8e267	32996bf10133391f0d6b49fa101495d0
32b04c3efcad380eac5e47af09d86937	3300309f0709837d0e6ae2854a39098b

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
33122a5200ff5bab1e5881d5c295c460	3404d4a68ec2082be814d9c80a6a22a2
348dbff2593df965f068aef6b2b6b413	348eaf73445326b0b37538f00e4ee0b2
34f8be2690fe183de0f87af9d01022a3	351956d8e5106c081578a571fd356e0b
351f0fe5c039917c0d5a74091998e57a	35536d66121709a9e6602380582f8be7
356b12f98e7252629947de9f589c3012	358b57a19413088874df6f6b8369f07d
35a493467338537f3060a5f915921f6b	35aba21c5f039c225e828453b44bc176
35b24360817300c5d10de016902f36a6	35f4a67d05b74a73d915a89d6a278514
361e591bb738480b0a5eb1864c803baa	36281406e48e64684b49701ebb4f35e5
368f43d3b2d68a5c1d44885c77bcb338	36eb62f3c873098cb488dfcb179eba3c
3705ce6f473e0d9e6350a371df46cd79	372a8c167b23813cc7ce63e9edbf64e6
37435ddc3ff4a1b3d76139bf2ff2a76e	3757b1abdebdf770e0213ef5f5495903
37a442aa2d9b63d85fe347b07b78110b	37aeb77c8b692565246f54fb7f2b5e44
37d0288f3538ec2012dba2c9bb6bec1d	37d65dc2f6ef7875a35cb7fd4575657e
37f523b7fcce4f147794cdc58e8feea3	3816c598720a93d8135cbe4ba530de37
381e952d81c9e18f7bf71c719629bc91	38eab493be24864abd4f65d93cb47d50
38fd07e154564aefdcd90891ac2da03c	391eb8239cde073b48f0c8afe32eaa76
398aa52575995a05003e696d30469a08	39bb07b4ead8b860d59086ee2020f4f7
3a152ef073000d460d9264d60f878123	3a190241881976ff5de19cd6c0cafcd0
3a717a1531ab1bab3da31d936efdd6c6	3abfc9971b0244bd8b3fd71bae538450
3b73aae72c644a583165345b1399bdeb	3b8bdc98c5f68f2515f46cb4e4e4ac0f
3b92e27b5ef6a539a37e857c03fb7e4d	3c2208941f2143faecd139598f7f600e
3c4c5b0218ccda6d928f5f17716b0e7a	3cb8c0ef9f090031a0102ea83b6746f6
3d0ad91352862096e1ec32fa9d263ee4	3d817f455e399ff0697bee202ee4624a
3df41d5755cdf49d0e4f3c7013b60f33	3e52728bd5477a5a55be4a0acf7d179b
3eac43d6d2823e96ec667e1b9aab6a73	3f041d0879df5f0a9011c247388462ca
3f9b091c227c556b625a112ae350e18b	405be7e2fb88f903abb9794937af3271
4079940929bcafeb62f2c77c8b107d53	409497aad8478b1f108d2c5dbce033ab
412f88a68e7f55ada514409a5a91ad07	415b1760a16e726ed41d51dbeaa4838f
4169ee85761b8249842e4ac3600643fa	41e7a67d6c7db42da0b2003052370054
41f5d44dd76b28af30b99f870d4cae0e	4230892c9a8a71ca5c16cf9648a87862
425faea3000b09d41cae5dab7b2f1a88	4289d22df5d1cc0954855cf2deb8334c
42caf927592375aa9d333f7178c8b690	42d5853289447d2c305ca6bc2c8cae65
43fb81c8a21da02c255b52226b38e69d	440f402a1d01e00e432b006f19d53984
447139f9f9ccbdc74b20b9bf529188b4	44a15e0eb1500f8dc5735e2f4df3afaa
450baba3ab2d45f36367c8d4b126959e	45265b9acb1185375d25c31a450df367
45325a7ecb100ead34b1118820650dab	4560fad15f456e351235ebe0368903d5
456f219ffd2e37df764355731746352b	4591d01a291b700efbc5b263c67a266c
4625556d1816142a1f8250bed15a834e	465dcf7b76fe5f096c8dedfa0cb07b98
4662c55431945f4fe0a09126aef5954b	4675b57fad56fd88bf34de076a7f39a0
46ab2971fe0c350486423b576a02f867	472f8afad528637170f19b23349c2a0a
473817b99403435b882f15f1c2eee92a	4759ec5c0a3723a44042a9a96ece3632
47be408472d47d7be5eb752e8513e76b	47e20a7f61eab0c70674b0ddf4e88979
482f683d8dab749cc3960421effdc7cb	48438d89ff24114b997de4d755216d3a
487abbadd74e843ddaa0da3af36769e5	4890ddcfd7270850711d8a00e8882f33
48bc80580308e62815fe2772f8f19698	48ecf8b999503ddbc10cfa534c2a343c
48fe83acfbd761b8a58e452e64bb1905	491a1c28a538a1fca31c39cd1e97cff0
4970b1306b05839117f41ad8e89af98f	49e7c66386b012f11cda865fe1b2b006
4a06811ba1b7ddee6183b96b0ecee352	4a07b8971b8b9a915ec5e2d77a6275dc

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
4a45262f75e06d4274c13d84d95214a7	4abeff74ef244747bfd002d6679ea025
4ac034bd7aec18af141c983ef6680c29	4aee8b48db04c5fe06284ef5da3e7239
4b428890778717c635f4fdf37e0c476e	4b882f26874fffd4caa6d768c1cbae9b
4bb936cef907ff44e0fd02b38b59b21f	4bc75c43ca502fc6251b08939be62941
4bf7765cda5fe17a1a14223cb379b44d	4c6a970497053090a98963bd0fea5431
4c8401f098965da00884231dd3460eb8	4cea477a5ed4fef9bcfa4d940e64a681
4cef2e575f765f9d2e847cee1af02b28	4cfbeb83f8dde134367709978db7201c
4d796194d32a6beeebeab0c96159602d	4dce51ac7341f854ade157f355af61ad
4e024fc6077966c9fb8f801e2458705c	4e0c0f6f721542d75b0fab90a1aebf2e
4e10b9cc5f9e32a4c782910b3cefe8e8	4e55869c4e3c6b1b5f58c10d4374bde9
4e80c948a46555705242b8ee562c46c8	4ebf42aadd1250e0e7872f090e9b1746
4f0c3021fa48a97d895e55f5d5c52806	4f7c4e13b5b80d5d5f63012440a5f050
4f9af5e1e7c6280b51e31eba35f654ab	4f9ea8df7af1ff43192fc06a8d8dfd71
4fef7c78f2336481091880daa527bd53	505db1b1035e0fae5de4960923c479b2
5085bf6efd8c1a0d8ae451cb98ceb589	50b28f7d71f5bc9e737e835ea1ce31b4
51047b312858c836bf8ed26481227f76	51e12adc70d90dd1827de80e979cc7ca
528a0d36fc6cc0d9d6c5911490795694	52afc1ef9c791c9b8370a57e71412239
532433a4c7b7017a48eb3e3469bf5a82	533425012942babb54542e61da671ce0
536bae7366943b3646027e1d5eec9cb1	536c4eabcefab22767da35676d712e6b
537bdf5599a0cd626ce23cb8b88942c2	5395f4b2998466929ed2a1ef6e2d7d57
53ce61071aa8a3d370324c0e4dd61466	53cfd02336c3a3176a5cc684fbac9b50
542d64cca238a631f9de90638860b664	54b1541096563b135ce7d49c68758b67
54b513ceacc56a011b9f523f196c4c46	54b63721c3487153864ce4b2e14892e2
54e04e505f40ba95c5ee2a340e331b5e	55234c043067e04a8303756a151f71b1
555b488d3a0cd0b4a96c98da59958c90	55bb3e16ce3651523ffddbb80556f922
55da827a2e1e53de9a99a5a7be8e6e80	5623bffc4a7c27a52499d4df91782aab
562b8e74dc5057c94db988dd1459cf1d	563d3e8d86a40ae3667d69ca4f0c61ca
56efb8733b3c014357a425faa76d07ec	5754c50403653d1e0a9c5a78750c7f53
582a0939db7ccaec722b695b82a71f15	583854135f05ec762885453dcbcfb258
58804f544d9e46c37e827568d5c35e86	58923e5e041047bea09ebde52615e46f
58d2d80edd037c580eb295071bef7f64	58e249abfd86fb07616372720cd9d392
592a3be9058c142c49ce8d369988a247	59481c38f59246062ca61d517a54e007
59e796eb656a8eeaef04ffc056d50343	5a01fb04c2d8da48e396883872221063
5a52b06836d60bae062ac4dcb9443134	5a63842f2f8a028c6fdd0b25fbd155b3
5b4a3cefa671039761fa46685a3ffeae	5b69fb3ed4b2343067462fe275b6a417
5badcba70a209a8902f65fde60f7d268	5bcf90204c2ea4e7404f0ad7fedd711f
5bd34fdfa77eac6ff1b1e1d7bec0f855	5c8636cf007ffff178c6c758395be859
5c8846befc08e672ca2997f443819d07	5cf8435ac2789e4c45acaba573719356
5d4df67fbc367497180305e7052007ba	5d77fe1694451b676a3355d850b6fb8f
5daca16e8ddda624b4066844053ce2c0	5db47a927bd78a49edcdf8b89b5be062
5de5d62a5e9408990eb5c6cdaaee0c44	5debe91e20fe06cd62a42a89a7464e77
5df31e5ab1f18abb6988c6a1baaf565b	5e682fcf7b7aae18780c6708a38339b9
5e99cbdbae0367d1582feb6b3545da84	5ed2ad24d08f6de8707be62dac8b87d5
5f6d0846dc2509936aa97afc0f2fed14	5f96ba85212f512d3a831e269eaa1a2a
5f992dcdf94a3ae1a19afb2d685410e7	602034bb7ebd33838d8d1dc69a25d59c
6090f8064149df3c1312e3480908fa76	60a9cdb30d576658bff5bcc22d423ebf
6129635ef5b27365f74cacfbac5c3a7e	63432acbe243613b85e2396e6f1dab5b
636c78cad73733d6b8b5d0f429a1bd65	63b22c816e9fc48c58bc564300bece7b

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
642cfdcac70e9a7f26d478e891ccbdfa	642e0166f03f3353fd1a112507c65b81
64b816007c77d52ed709b39d9068ee31	659c6c75bc33eb082d190f7ec3d7dc88
65a7354887d20d2e5390758e4ab22327	66cf4157605d7c274fac57f382c81c50
6702c668b5a7915040f29799f0bff9f8	676df385a8867b88b70008d5fde5a029
678a52cc3c8ff5de68650ca429ccb39e	678c155ab6ded113732106f1e77594f1
67c4d827a9938fb603e9554b9053ba93	68f15dd40a1b77560ba6a2140328949b
6902a4fce7c6d5541cfedd925d887489	6986c2febf0f4881874c149ed845fc88
69afb814ffd217c8cff57a8b0fd099e1	6a6e781b430e96c2a68ddf069774f590
6a84f185b902b8fc97c139f096c1fd33	6ad76f2b1b2bbede5c9e2aa481100f97
6ae6862863b405a4cacc0d4710de0d00	6b104f777ace354e32f1f0ad86e4d622
6b26b953fc279e0883d5ee11354b8661	6bd0085d57593d91bf2c1987143192c4
6d0e467d2bb49bfd9a56eaa726848bc4	6d2452e526c51e90810ab76ffc67ade4
6dd910e051b6baaa857af6faa33bbb90	6de9a01aabe01cada53168a6046dac3a
6e2d6d499c0fef5e03d66368abfb4b3b	6e6ada214d40a58c73a40d53a93e31ad
6ef7f933c16b5a88bbcdf5b45fdbcaa1	6efb83de57620f423d797bffccc174fb
6f1cc4ecf8dc7edb512cff16cbd7cb9e	6f6d555dc1b403889ac83d2f10538589
6fc73251c49be187fed376cdff1fb521	6fdfdffeb4b1be2d0036bac49cb0d590
6ff72efba531ebff2cc701a67bb7d2b0	709e6924c2a02282fef00663bd1f7ad8
70d6b3d2aad144c5960adcf167662277	7112c8c3ca8753ee29dad31dfc17c247
71e764948548141d9a3404ca62422696	721c13f0826e0ef081e96e67052d58a7
724bc114b01c5d660b95dd34d6932190	728b67ca6196ad196a0045b2da1755b0
729b4e4215add22f0a9f46666010b2c4	73555509028ef8d62f50b1a57ad3c809
735616922c8f6d12c77fe8380cc45c63	73a2907611ee9ae8fb91f1c5295e10c9
73cb1fb669dae3e830b4c3ef800750db	7460a492eec81d3ff7d0b9d0cda7ad98
74667ccb3e7f090d565cfeba0c1dc480	7466cd69198c0b58ac8a9e0aef095828
749cf4eb854bc589101f15bcb3b9e820	74a9d2afe4c02a48f68b7ae8c1bedbbf
74b70f99af35965cedb6e443a40ee85f	74c015b3d73bec0046fdb53b5199e286
751070e0d3b13c818116cb756e9c8739	7521cc7ed62795b4b317cba652358e88
753455178be69c892ad555cf1d635e3b	7626b255965e014a33f0cd51240c28d4
764e2334fc93270fb0b6085e6d98860c	766928cfde67f943fc4498d48a9c84ea
77a1573086a94a6204302a6c4a402e8e	7800b7642d557b785800e1c482d68066
787bdd1de6c524a6ec49041560b68138	78d03e55a867428e3a678f804f1d871c
78eaa0b308025bfd70036e5fa2cb6fee	78f8a28871449eee80df24df4d09bf74
7902f3110fe867e6949f42cf45cba443	7926a3a1708da681d54dd1a6ea45be37
7970c24a511a6e6ae6b7c21a2deb371e	79932e306e412b725e7f30cb26b77943
79b85b30ea285520897bd877b4344b1a	79d3310ec654f485ab78ece9abb9dfb0
7a1c2c9cdd7b0d68f1f0925edfae401c	7a1fcc3046dcc5daaa4ce5f6abb0566d
7a598e26d7959b528e9a7a875303d6ad	7a9c74897b08ec343779f0e10e5cfd65
7b1d27d1ea4da6f2f489c151fac41590	7b32f51e6b00947c33bd7b5a3bccc882
7b961b72e35a535de5f2f3b9ec11da15	7c2638759f661e5ae8f50f641d9e21ca
7cf887cfa2fcd898626a284684851b81	7d426e2827126392a664058f8ac7407b
7d4cacd8f627840c7bc8e5aa6b3130c5	7dc39daa8f8ba620503effa9a756649f
7dc51c42a4c864c9dcd3597e4016b19b	7e0ea94c38643e3fba3dc63042f38b06
7edf08d433c160770d5d9dd4ecaa1680	7f4b30daaa62a00fabb9d92a4dacbe58
7f882517f15b9d2e9d6d3d01bc25b707	7fe9df44100806be5e9c676e68fce782
7ff4438c54f7abc5ee61da0b14028f8f	807767735a05a4f27b65a4b349278622
8080c1ee8d67dc36575bf4beddb50e66	80c784392c2890805d0592a914b3899b
80d87ad5b6c761b2a6bba8b4a5801903	810cfd1328811e211d7d43a1ab2a936c

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
8129798d6b993fbddb06b557501096fd	81a2868c4858e00462d2937398069068
821e4c194c721f6f40b6a63b71229677	82591cb671019951c9afa493e8bd81cf
828141c315099a62c97d8dfd814a5350	829398cf7e84ef7f645863022acd2705
82f792a1a4abab93058f916e70283252	8388c2abd789563f9762c9d7beab4140
83f306dad45143effc639943d8d5bbe3	83f88691bd0a571bc65c971e153877c2
8452649012c7f2546c71f71518a06812	8461295b201f4583540638oaf587c1bc
849bc79b15efc6e0e552a1dcf9635ee8	84b153c0f074e4286be1bd704c20cd50
84cfe7687705b1cbc1c2a2eafc9c6636	8508de9e731b1b53e44c6f3339b85116
85b499d1b73b7c4728c577410b47b752	85b8504e8c075d2a1cb20319dbde1ece
86094f463472dab508385b1f16a4797b	8629fc041466b6b51fc13d6e9a293d12
8726f8d6889474aabe8e6fa235415f75	87ca29e17722011cf27d54e57065f8a5
87e7443b671b0422caa6aa81175f28c9	88078f889ac9990889753ca594416170
882701961690b3c93913228895224788	886b8028ad216be965cdb00cd69b21fc
88d67a43d0a595f20e91e021b94b59fa	88ebfe19f13b2663e242a2421e5498be
89045d06eb6c0a45e47bc00658251699	89a6428bb7e1174e9e4e8ac99f7b7af8
89aa9cf40eea5af42a3f98e606990f1c	89de2036b5c82a442cc2451850247d4f
8a45306096c9495a2d0b04674a1901c7	8a67923a4b152305bbdedde79db5da5d
8a7bdb50251b559e715a4dc93c28b060	8aa4d18f9a19c281b8eae72a7bdceda9
8ab94d7a4d4429777a7799b00f9bd531	8b79567a25204f5c4160940fc62cd19e
8b8a263fd021bd5d1496031e711e8727	8b967007497fcd46f3cb703a70df6b7c
8bbddc0197e41c61a1397aef39a79d25	8c651ef402aece272d5b9d99f7cfab73
8d1551ce12859b97b4106a3c195aec4a	8dc665525e8c08937f7ee0ae977abbfb
8df99949944269e3185005d0d175c244	8e0059ab10294517c433d8ee58a52347
8e01e390bbbe011cd06b71fc01e69f43	8e371ef40e97f4e7a9310339300864b0
8e5432b455b945cdbb4d81a7da4d292d	8ee8b6db5c0dfecd0b052d39bad8d503
8ef2cca3b6380c097cb81d5ede5b0d8b	8f09aa0b381297123fd2d29cba0ef57b
8f91fe881fe19704b40f135f3a9bd930	9077ed6af5b65705fc4e65b58f4547ac
90825b4778d2f4f90cd3e5174a163162	90aa2cc39e10ce43f7f31e90961f0b5e
911a14e0c36f55a364e7798f576bbe2d	9129f199be8ca182a541336bbaaeafb0
9143e250a9c2f3d0d0636151c1cd997c	91cb657385df8f51876f5e1fd88f0352
92e8bad1cbccf8024761dcf7dd154e86	9335b6b91fbf6b7d72fa0ee18ee16795
93c19feed3dc74df766f41ec67c5e9b4	93fff21615665e21817ccb6e94f7eca8
941f3287588ec29e5837b0d491997d85	948f810bc2101d8ded70226c91726e72
94c0972b06c75456ed574dd46417b1d8	95fee6047fe07b93ad4ad6ebb2343abb
96163131518860046045 8d44141364e5	96a7f60ef339d586b35754cb8106dd86
96b30e3c9fb8fccf794a7ba40c6051c0	9715bbe0c4f594da9bbd99d2887f2061
973eb04047271413b719e2518e421362	9768289d64c7f08b4e2bdad7c7fe014b
9778598d05904172cd4b45565c1a3f82	9789089b434b5d4f23b7c7aaee61fadc
980df59d789e70e3744d811cd3528904	981e82f907d1943f3ee06e05aecf7c31
985441750ce4c52e4f72d2bdac629d8b	987fef059b1de2791ae99876c864d26b
9881b40faf335c3424660e996d003ea5	988f08c71b487bb22b9228638056f698
9898a0d14d3e18117e86e1e90f075efb	9900a18be22e3cea0c515d198159d22c
9a3375be7f530447efdbe21d5ac4c683	9a4707222ec9730a7cca2ea4e7dda688
9a601627690a1963cf95c6a361a63a6d	9a87b59dabf265d46325caf5056ce103
9aba0491c23dbc4c1de4efc7c74ed2f4	9b5f7921acf75bb8c816c64d95ac5402
9bc194ee89c100daeaeeb5e7b6eafaf9	9bccb77b40407891eacb940ac606e3dc
9c2cdc8a4f83f4a4e1af1c5e93fdb169	9c404b8462e525393bf43f9faec04964
9c6f3f69163133fb8e56ac4a6e163452	9d045fb3df82c2296f3934f045dddebe

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
9d1c1b63633698e6cd6d9a9e4fc0415d	9d7c3389efcb75fc3f08bc8e289256e0
9db1743f4d0578531fd929063372c6a5	9e38bb2fad42a4fff7084e2e22768de8
9e4cccd5b22c6d6eca552c4008a46d5e	9e7a09776b0fcda37f3d232325e71422
9f148789adacb0c97358c4ed010c9b8d	9f43af191437cc10ae78fb9a924fea47
9f69d25294f319783b283d4b946a869c	9fb83f7dc4c4fad27922c25caeea6677
a014a59ab22a459d4a95a914502b157e	a0232db9969b9aa5ee85397a90a95e57
a03041f33d032c6550d6f78712c8153b	a09d5b3376a8fddb03667c7896a6ab40
a10fe5fa3f861d6f3faf705013e43045	a1233745dc77c2ca801bc3a97c6a7ffc
a231dba1d6e47f7664e4299836f0277f	a27d0483015c0c03c63d72ef35bb8cbc
a2c8a63a043cfcadbc334066bf597ad0	a2edb82bbef539df5ad7201b9e290208
a304e76bfca925d521934c67c3f2648a	a37dba2ba1d9e0b4be37e7ebf82579f5
a3805ab6b4df44000de89ae9998335ea	a3a823abd653691227bf8d35c88e2b0c
a3d0856589da38ef3acb65e0e06df850	a44366ea0764087efbfb21132fb96a65
a501120d70f03c123b02825dd61fc2c8	a538ed02f51a3ffd834ceb5e1fa4cf76
a555609372d8f3ecd324dcd216d7e5e8	a756fd2efa1e84718486d1b1b79a2243
a7a3dbf9dd16606262d647c7d5814be1	a7c33bf6cbd0079e4d07d9ead4453ed6
a7d14e6b8b3f2fe2ae870bd7b166dbd1	a81d7d49b08430c7e113938fe8a39059
a84ff0576453a0df594e47dfe0b445dd	a8681b151753d0c40fb8506c4bcaa5f3
a878ba26000edaac5c98eff4432723b3	a9002a1a2b9bf1082b5d7971887a7f2e
a90a54ee776288220f6f3aaed42a261f	a94a4ad349fbbae6a85836f9124a5720
a94e97bf0a80b715ed60daa627b29b2d	a98e7bc2ffad6d49f152686f96788dc3
a9ac8ba20b970d019ba77afed8cf50f9	a9cf441025a55b22bb56f5dbc32074ab
a9e662104e3954c8bb0b02c6dae0e3f3	aa47f80e669d593af7aca3757ffc8203
aa5faa049c93871b67283355a9acc401	aa66c8087c05c7f45601d6243ac98b0c
aa6c72571a71989e5c973a42c043d8e1	aacb735438150b870fc8e3acb0fa7745
aba2a43121c8e22d2ad43afbeecc9815	abe30af73dc9139bb15f416eeb77d2a1
ac15e6623ad86c11a714a50a04674aef	ac5ff80a85123df0d49056032bee3178
ac6ceb302e40c0ddb9170b5c6614ec0d	ad07beae6755922e6053427bce7d9ed0
ad11a0bc29f205436c892b3f9a1e5070	ad16f818470d09beb9757b873d3feb27
ad7bffe4fde25c9a4e6b72e4b69bb9d5	adac1e3ee33539259c0247c2ecd2b69e
ae6af24501520d8e9e069cf6e85fb87a	aec1554bd6d16a309187809d658cf34f
aec479b1c2c3f47a0fa24c93ca45f602	aedaf8bcb483b96fb498c41e90f678e4
aeeb4ffd2daeead3968c64a6fd7f9071	af34cdc7d8967e7633fdc1a0355b30e7
afd47151d0fb1dd6ff2f3502e8cb79ad	b00c6a1bdac52b817defafad27560bf3
b085420cf29be8eeda7c214c3fb12c04	b11cdaef763c2b3735749375c87da51b
b12e68c50f2729b72dcf9c7fcb9f65a6	b163c5000cd9e08c41982188f3f4c802
b173c0ffeedaa760a88f6322bd68644a	b1c59fefcda733c97c50092068cc110a
b22d79daf95796dc3b210132fbbec188	b245ed377508f01bf6a70dd2d550c6c5
b2e221eee320a6c7468dc8865f7b4ea6	b2e6f28df5466e9a37c60650ebed5e21
b40278658c74c98dc303da4eb6fbd838	b464459e265de6c10cc3f70491315b82
b4c7546afdbdeb8198569d250bb06500	b550ac758f5c3bae3a1113fa1d772780
b5d1bee5a22623b0419b2205467f8d6a	b6091e07815030444f8a9a777ffb72e5
b62d7a722e38f08684292926e57c2cdb	b65d46c0ebfdcb53b371fe651ca6f31b
b67a53fb73514d9ff24b185ce25d4f26	b6a4e2c5ae1baa851b8abbb0f6528a2f
b6c9fdb500ab54d53f5c59990d1a2297	b6e43cb2184baf4ca08086ff2264eec1
b799cee7ca88e32b23867ad61da498e0	b7b01831ab619765b1fb3589e4dd3565
b82e00dce4e8c479d1231007d39a040e	b8c3ef0497de79446a0bb4a8b242a8c2
b8d0d6278e30f2143062206027b947f9	b8e1be3732343d63cb8c4fd84228c9ca

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
b8f414b2c6b539f2cc310dd9513c4fd7	b8f8b8ed005e571de982ccfe70c36e42
b9c9af53dac38888bd07e3f5da5b9ccd	b9e7e85faaecdcb25ec0ac8478cc09a5
baed21297974b6adf3298585baa78691	bb3c3d56c936b77920f72bd9c4958cfc
bb8d8f3dc0bc261118b3e52d6850c0e7	bbac5c1813c43845344726e549ddb1b8
bc04730a1877c79299f3cc5124b9eb1d	bce0fec4b42d6964a88e097311bf655c
bd45fe39d359ac635fa87e209149a084	bd563a18e5fc36acad88599e16ef0cb8
bd9a2895d87ed60fc0017fd2213119ea	bda2c02b8ae5de607227459f60f5392c
bdbe835094406aa6ea837ad6b0d3b6c2	bdc31b71eb76a8356183a8716b3e036b
bdf46368154be6327f5830bf959d36cf	be713bb1f92715d30560e3932fcc06cb
bf5023ce4f49edc19b77ac972bf4742a	bfc3900c9b50dc8d63d48e8072399b2b
bfcd395ef32a6a4cb210dddf0f32b0d0	bff890000110d2407c98362f49054b5b
c0785f417aaf685af51c1212a0aa955c	c0c5b181c0f1220b05134f186b73449f
c0d0b41a38ec4e69bdd99bbbdcfada66	c0d0d27ad4403a31e24f674146cd619d
c1d81be7cb3cdcee745e5c6d07002e14	c206992f7c6836ec6a227a6e29ae7609
c297eee5ee9f8489519ca3d888e4c16b	c307b109901ca566d4244eb319d642de
c3d7a19824da82aa97056400b6b7ba38	c3e6a3ec71c952f1098389b7c3e2594e
c41d0febd6a9eae9eff5ee3f96d72cbd	c4489745d1d871523961995a9ba00246
c45a6ecc0e115ccafbdb624f60d291f8	c4828afdd0e365d9a699c83aa93f0137
c56d5e988b72166651cc925a0f203ce9	c5938b91184c188fb5ced8738bcfa2f3
c5aee1e60e4183e7eda5ea6fe53bc540	c6713b1c6db25dd962cbb4d23e8cd35a
c694832a7b0625ac90fd2cdc312fea89	c71929da3ea3da497f1f89199c77d1ac
c72238b422cf8c73589159aca65bc4e0	c729d202af55e981fe987dd3a2131bcc
c74d96e1d8181ce5bd9d435d54bf47ce	c78120aa5124274b458ebbdce5cd60ce
c7b338d97488367276f1bce4a6245cca	c7be67c63a9698c17230d369e36d5eff
c7cccfa2500b537adbedcce469fa3480	c80acb98bb2fc370d21877c215765e82
c8241996203399e271bb3591dbde255a	c86439ecd578a5878f99986275bcb785
c867554929d2cf5bf5b0453979bfaa6f	c8eba4713e9aa73517cb580667cdf06b
c96a9e3d8d1984cbdbcc208a8f65de0c	c9f28696fb0365abcc2adf61a2ffbeb5
ca76f21c17ac39166a9965f98ceb5a37	caa4f218a0ad6331e31aa948931b9c57
caa6f226d043938a3d1ea71dcbbedf18	cadb6eccee60be126c2725b561833c75
caef45377fba37f9839f89b87e09a51c	caf65a2e4dc715a2a77e2ddcc53b40f5
cb0ad75e725eff6ae358b7b32098f800	cb8c89fbb6f9066486d628efe3630809
cc4027dbfe32d73626da1ab41f34ce43	cc62c38670d0c89be0319d4b74e79947
cc69aae4c2ea987a1d718cb039bebfa2	cc755bfe842d44f14d87b77848e4ed6d
cccc302257610082f064ee7d743095b8	cd49f1e0f70cb74976fa741316081c9a
cd70eec9255ea47aca61a32921800fba	cdcc63adaa351be416b61da0dffb2c2c
cdd42d224cabb9b5455a660796e98b52	cdf33e1ef314e6928eac9ae9f6ff3660
cdf59503c968048b5a5359cdeb4c2d84	ce78f54b8409ae1ecce6f53a63a87bf5
ce8d8f47969e704a7e3602a9cb1536a4	cebfc2e11ef6e6155b42893a386066ed
cedbe40f5114a561da596afd24dabdf1	cf596eef42ab2e866825779b10380e66
cf68f6c8bb88d7d716863c187e8959af	d0034f9e2cbd4b1588d31fdcf4a8a8b3
d050737ba5783673142da45ff521987e	d0595f53a68e289e55c9aa37546c6c89
d0672bc0b23ed15ea25fdac808cab771	d0f1015c0aa6ef6ca260b807e452a311
d146984e4e4d33f6c9925c44649c732c	d158304f091f1e994120082cc5103e5d
d199468856457236221f132c8a222a1d	d20ad5c65e12d01fde9c5d332baee48d
d22791312dff3f12401bb1f2f37b5b87	d277417d04f5e8377b6d211679772364
d3367aef91417ee4991ed7680c0ca5df	d33e50c227aa01ec4d8d225144b8f6f9
d369380616c2d38b934f57eab8b706e2	d3c4a6a57c91bc8f54ebd945b6dd3437

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
d416917130eb38a3f47ebd351809578c	d41f06f5901cad65f6b3d06409095809
d485471c1f5da4caddcdaa9e06397933	d4dbf6e2f4cbacd647bcbff8ac4ed34e
d522acef1c11bc2b5d00fcf7fee5609d	d55ea67f328f0431e971317a8390b020
d5b6ddaa188fbd95cd14f50d69204b7e	d60f28c5414bebaaa358d14dd79bf8b3
d67f0e1fc1334e548d0a993200535ebe	d6cbe2f164dc0236df2dfac91d5bc961
d6d93848388714b90f16caf7e80ba6b5	d7013c912e48b10dcb651a52f87d7c27
d74f59d6986794a8d08f43a590aefd5a	d7994b8dc70c86682fb2c6d9df1307a7
d7c4cf80641fcf022c5e4fc9768cce00	d7d8cc5c1cfaae6a6dbb123d936a7610
d7d9a9cc107f9db783446e39bff8db09	d809535cb0d49a1d0aefdcb5794c7a09
d826f06397d887cb4e59f437295ce312	d86db1f18ab3e1d48f97212beeedd7c3
d8ccfeff07bc987441b96eac152809bc	d953180001a27c8d93ccd3956499802d
d9568a04f480050576d275af369d9c14	d95d22f3312cc34e4d23f5bef393d62b
d9c296ba93ffa14b70fc5d8eac458fb0	d9cc7c96d37030d9b3ee8c0a51137356
d9d2956dfd94cf2b59d60150da7578ae	d9e333a988eb7e147318d3b3e8ba4cb9
da906889ec48dd94ba4cd2bc0f94b3b5	dabf42a499293991b1d95fe6022341de
db286906dae31bd10511f9ecc53a0c78	db5088b2f8addb2956465305 80c86abe
dbc308fc61be6c071342e9678a65d788	dbdc638def1aa026556381dbfc365b2f
dc2f523754ac143dad541d64bc0b31ed	dc4389744f753fd5bf2b0e0f61047129
dcfa0b5305640443622edd8d7a983af4	dd2c55b030659ba383ee9bc5bf438f5e
dda16b2129a691051b49acd241c5465f	ddc81b7546bec1fb8abef356f5c2454f
de2f97d310faeb6470c3de93b5f58af0	de8112218bba2334fcbc5c1d400cb005
df3d9698ae3d2d19127d25ca35211971	df610829fe276fc3ac41a4a67fbfa0b9
df702cb209aea14d728e334cf80309b5	df99c7ad2b879d4e5e0842d118b429b5
dfc7ad1a64c8c9a54dba25395cdd6a7f	dffac79ef676d4ab0f0791575dde37b4
e03881219a1cae25cbdcffc319fb129d	e0697461cd6961ea62daf1571e68bfca
e0f6ef60c2d34dd49042ed5b287ce087	e109836a2e7146ac1cf54f62800563c1
e1615caacbc9bc9f332735ac21c5a037	e1b435a2b0d201149fc4a2be883dc319
e1c3b0474914f52381b054c8fef9e140	e1ce3da256b2654cdfdf03d6b4be177f
e27eb6cc5ad18ace1c1591026a368bfe	e2c82a0891c23d5afc86cfd6115e6b7c
e35ec35ffe86238a3a7b99851f9fb084	e35edb8ad7b18dd38256ec6f0360b7ad
e3baaf38a44fe84445edf5fdfc6f5339	e3d80c62ea1b9395b7fa369d70889a2d
e3f4c8f58bad76531f012cc9e2b2e25e	e413393560638c6ff4e6dfde531ecbee
e423de9d506d6bd964aa57ce9f239ea1	e4b82b59b52787f2b7fdff6fc6518bc4
e542190dc5058a8902b217e46edc88ff	e564a2643af6840734ce5a7ea1e93179
e5a481d09e735747f8c46d7df92b32e1	e5b5cf460953ed11f006153941a6cc9b
e5e8c17801f7c27d506e0906ab734ec7	e621488cc9863e368c1b765a609b1e80
e6e612aa05da6a2cf4f1caf485518fe5	e6e7d456512f492dd78b905d6ce2a133
e6eb3eb37df62da9813c75d43a1fcb8f	e714ac071c6bb3853ae6e61720bb8ae1
e7e5cf3698683455744529eba5a358a2	e89c3063a53479bed27324ac5f420a5a
e8a8c5877f0512d0728f1262a0b47314	e8ad005205a3b7a52bfa73134915114a
e8cbc216cc2edafca5825d2d65054cc5	e93f4cdd1d173cf2886bebca186cf821
e97f8501805f6df0db440a4ac3af06e4	ea7a606cace4fb3e16c1664c8acabc9e
ea9ec94611d790063b3d48427af837a9	eaa570561523f1759bab32c85f9ae267
eb1a16854915bf5d3d10f0f22ee9d237	eb40fe2dc7178a07dc52f24390a575e9
eb4c0e8744ed09671af7d8373c717efc	ec5a5b4420810494c18367bc97a4a8b5
ed42fb2c20b2b366995a812ad466be59	ed4e5953c74f95c1250337e4a700d438
ed8721d5865f2393cfff18f7ef18895c	ed988cbcf5a73dcd1d4fea277635a3f6
edb7c7f26adec4bd34e890673d0dbfab	edf1aa187f3f47fe6b44d0b17097568e

Continued on next page

Table 8 – continued from previous page

MD5 hashes	
ee083fc36481b93367c87f818ce903a4	ee2f7450e1d3e9e25d50cd8e623a53f2
ef7947f659f74e2b5a1ed6b8b367cd46	ef8b1fa39882889a44ef2e41b4270158
efe183c9a23b96f321f235e87717a4b8	f0193f89eb2d7505e9af1ea444585304
f01f523dcae2960898d68b811b8f3558	f0354d733bc57021594dba4c16194320
f04d843bd0d36aeab213aeef86553adb	f0cb9c2245f039c56c3339453e8dc868
f104c1cdb772b8f2ff5c9d7cf7db6267	f10b77de13fff9bd80281526f0d1b7e6
f196e2d85eb00c87dc2461ca85846d35	f1b181aeea9d09c510e641b5599adb1d
f26518fa9e4404333a3163904723c17a	f308d8bdcc6dfbd79dc95676553fb2a2
f33a6e7a62700f495072d38d23e2f131	f3cdb0e349a7388996039daa3aec1b17
f481dacd53a72b4f7a9405068c0408c5	f4b8fad139e06829bffbcfe0fb85d45f
f531a20326f16ea9a1667c02970e8798	f57be4d55d95fee56d892c8acf82a4d1
f58f6b22f6cdd4228fe4c987cacaaba8	f5a3a06c99e01b856e55b3a178cedd51
f62dd9ad2b95a4a77b4da42c01052a03	f6343a86cf40def075a94f4145b4cca9
f65abbafa86ddf2249074c9fcc4eec98	f6c9f9a5e791306a9a23c50fabfd9257
f6d61302d769fb29d380435e4f6e0edb	f6f8360523c986ad759ee7cfd1b15d2f
f74f63be63d4caa0a46249f461285bcf	f7d17be815abf0c384b0969bebff26fa
f7e02bd8390984ae14dd6cb1362a9881	f84c09d2f84e993ab05762a513b1021d
f97fcc229d20bae904c8f12cc7fe9aae	f99bc9b65c058bdd470bda7c7bf6de80
faace938224be13f0e4a61353086b21c	fae62b1e0ac190d25084d8d8ab70e358
fb2ac457078e986b0fb27355e783bdd1	fc620d986b44c666d4fa1dca3671dca7
fcaf47a1d4dc8dde3f35ecfd4ace9962	fd22b70257316b2863f07a36bffc5d8f
fd736f06d95b164c50a996f27d23265f	fdbd4b86bd358188e90da24f17e17eb6
fdbde2e1fb4d183cee684e7b9819bc13	fe13da4349c2b8e6bd4f381a77739812
fe9a74f637d72cf1aad54409f4777a78	fec6d5047337d0c926415f741f63bb8e
ff106baf4e1c35ec2796ff930264f750	ff11067e2ad7731e41f89896ebe44a0f
ffd51c078232fbd9b3b507b43bfe72c8	ffdcb4f37374c7b4b26cacf838b26c56
fff09529e2bb5e7dc7cc4250c8b80613	

B DISSECTION OF DOMAIN GENERATING ALGORITHM

Listing 3: Dissection of Domain Generating Algorithm used by Conficker A. Source: [53].

```
void sub_generate_domains() {

  GetSystemTime((struct _SYSTEMTIME *)&SystemTime);

  if (!( SystemTime > 2008 || month > 11 || day > 25 ))
  return;

  seed_random_gen();
  get_time_from_popular_site();
  succesful_download = 0;

  for (int ctr=0; ctr < 250; ctr++) {
    prefix = GlobalAlloc(64, 32);
    domains[ctr] = prefix;
    length = PRNG() % 4 + 8; //range 5-11

    for(int i=0; i < length; i++) {
      prefix[i] = abs(PRNG()) % 26 + 'a';
    }

    prefix[length] = 0;
    strcat(prefix, TLDs_array[PRNG() % 5]);
  }
```

C SCRIPT FOR EXECUTING MALWARE

Listing 4: The script executed to generate the data set.

```
#!/usr/bin/python
# This script runs the malware on a KVM machine.
# The script will follow this order for each malware sample:
# 1. START malware <malware> on <datetime> (log)
# 2. EXECUTE malware on workstation
# 3. WAIT for x minutes
# 4. KVM_DESTROY workstation
# 5. STOP malware <malware> on <datetime> (log)
# 5. LVM merge clean snapshot
# 6. KVM_START workstation

from datetime import datetime
from time import sleep
import subprocess

# Declare variables here
starttime = datetime.now()
logfile_name = "malware%s.log" % starttime
malwarelist_name = "malware.list"
minutes = 3
DEBUG = False

# Some help-functions
def log(msg):
    """ Prints msg to the log and the stdout """
    logfile.write("%s\n" % msg)
    print("%s" % msg)

# Open the log file and set the start.
logfile = open(logfile_name, 'a')
log("Started script on %s" % datetime.now())

# Open the malware list and start the for-loop
malwarelist = open(malwarelist_name, 'r')

lines = [line.strip() for line in malwarelist]
num = 0

for malwarename in lines:
    num += 1
    log("START malware %s name %s on timestamp %s datetime %s"
        % (num, malwarename, datetime.now().strftime("%s"),
        datetime.now()))

    log("EXECUTE malware %s on workstation" % malwarename)
    if DEBUG:
```

```
            subprocess.Popen(["ssh", "-p", "2222", "
                Administrator@192.168.1.2", "echo 1"])
        else:
            subprocess.Popen(["ssh", "-p", "2222", "
                Administrator@192.168.1.2", "C:\malware\%s" %
                malwarename])

        log("WAITING")
        if DEBUG:
            sleep(minutes) # sleep <minutes> seconds (for debug)
        else:
            sleep(60*minutes) # actually sleep <minutes> minutes

        log("KVM_DESTROY workstation")
        subprocess.call(["virsh", "destroy", "workstation"])

        log("STOP malware %s name %s on timestamp %s datetime %s"
            % (num, malwarename, datetime.now().strftime("%s"),
            datetime.now()))

        log("MERGING LVM clean snapshot on workstation")
        subprocess.call(["lvconvert", "--merge", "/dev/ewi1439/
            workstationcleansnap"])

        log("LVM_SNAPSHOT")
        subprocess.call(["lvcreate", "--size", "5G", "-s", "-n", "
            workstationcleansnap", "/dev/ewi1439/workstation"])

        log("KVM_START workstation")
        subprocess.call(["virsh", "start", "workstation"])
        # Wait for start
        while (0 != subprocess.call(["ssh", "-p", "2222", "
            Administrator@192.168.1.2", "-o", "ConnectTimeout=1",
            "echo 1"])):
            log("KVM_WAIT for start")
            sleep(5)
        log("KVM_STARTED")

        log("")

        if DEBUG and num == 1: # Run num times.
            break

malwarelist.close()
log("Malwarelist closed. Done, shutting down.")
logfile.close()

# vim:set sts=4 sw=4 ts=4 ai et:
```

D
LVM AND KVM SETUP

Snapshots in *LVM* work with modification tables. The original Logical Volume (LV) keeps writing the data, but from the moment a snapshot is made, the snapshot volume also keeps track of every change to the original LV. When a *merge* (revert) of a snapshot is requested, the changes that are in the snapshot volume will be reverted in the original LV. This changes the state of the original LV back to the state that is was in at the moment the snapshot was created.

A Kernal Virtual Machine (*KVM*) guest can be assigned an LV as hard disk. On our test system, there are six LVs present: the root filesystem and swap of the host system, three for the *KVM* guests, and one for the snapshot of the workstation. *KVM* is a virtualisation tool which works like *Xen*, *VirtualBox*, and *VMWare*. The hypervisor is a software package called *QEMU*. An overview of the disk division of the test system is in Figure 6 and the the *KVM* overview is in Figure 7.

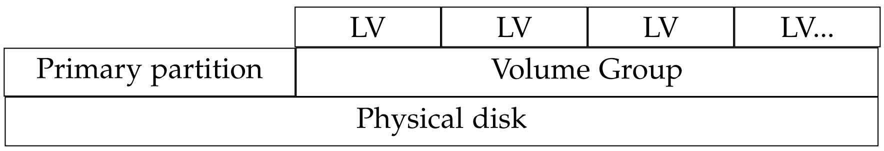

Figure 6: The *LVM* setup used in our measurements.

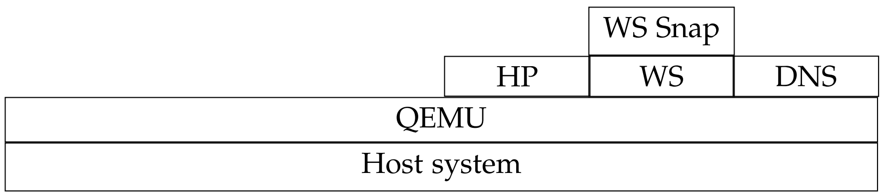

Figure 7: The *KVM* setup used in our measurements.