Cerberus: Detection and Characterization of Automatically-Generated Malicious Domains

BY

EDOARDO COLOMBO
Laurea Triennale (BS) Computer Engineering, Politecnico di Milano, Italy, 2011
Laurea Magistrale (MS) Computer Engineering, Politecnico di Milano, Milano, Italy, 2014

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2014

Chicago, Illinois

To *Etta* and *Nino*.

ACKNOWLEDGMENTS

My first thank you goes to my parents, Laura and Carlo, who have always encouraged me to follow my dreams and my passions. They have supported me during my whole life, especially during these five years at the Politecnico: They have always decided to sacrifice something of theirs to make my life better. Among other things, I would like to especially thank them as they allowed me to study at the University of Illinois at Chicago, one of the greatest experiences of my life.
I love you.

I would like to thank Prof. Stefano Zanero, Prof. Lorenzo Cavallaro and Prof. Federico Maggi for helping me throughout my research efforts. They are the most professional and passionate professors and researchers I have ever met, and it was a true pleasure to work together. A special thank you to Prof. Stefano Zanero who has allowed me to go as a visiting student to the Royal Holloway University of London, and to Prof. Lorenzo Cavallaro who hosted me and made me feel more than welcome.

I would also like to thank my friends, who have supported me, and the people at the NECST Lab, where I spent endless hours working with them. It is a really nice environment for research, where labor and fun are harmonically mixed.

EC

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

SUMMARY

Botnets are networks of infected machines (the *bots*) controlled by an external entity, the *botmaster*, who uses this infrastructure to carry out malicious activities, e.g., spamming and Distributed Denial of Service. The Command and Control Server (C&C) is the machine employed by the botmaster to dispatch orders to and gather data from the bots, and the communication is established through a variety of distributed or centralized protocols, which can vary from botnet to botnet. In the case of DGA-based botnets, a Domain Generation Algorithm (DGA) is used to find the *rendezvous* point between the *bots* and the *botmaster*. Botnets represent one of the most widespread and dangerous threats on the Internet and therefore it is natural that researchers from both the industry and the academia are striving to mitigate this phenomenon. The mitigation of a botnet is a topic widely covered in literature, where we find many works that propose approaches for its detection. Still, all of these systems suffer from the major shortcomings of either using a supervised approach, which means that the system needs some *a priori* knowledge, or leveraging DNS data containing information on the infected machines, which leads to issues related to the users' privacy and the deployment of such systems.

We propose CERBERUS, an automated system based on machine learning, capable to automatically discover new botnets and use this knowledge to detect and characterize malicious activities. CERBERUS analyzes passive DNS data, free of any privacy issues, which allows the system to be easily deployable, and uses an unsupervised approach, i.e., CERBERUS needs no *a priori* knowledge. In fact the system applies a series of filters to discard legitimate domains while keeping domains generated by AGDs and likely to be malicious. Then, CERBERUS keeps record of the activity related to the IP addresses of those domains, and, after Δ time, it is able to isolate clusters of domains belonging to the same malicious activity. This knowledge is later used to train a classifier that will analyze new DNS data for detection.

We tested our system in the wild by analyzing one week of real passive DNS data. CERBERUS was able to detect 47 new clusters of malicious activities: Well known botnets as `Jadtre`, `Sality` and `Palevo` were found among the others. Moreover the tests we ran on the classifier showed an overall accuracy of 93%, proving the effectiveness of the system.

CHAPTER 1

INTRODUCTION

Botnets are one of the most widespread threats on the Internet. Analysts from the industry and security researchers are both striving to mitigate such ever increasing phenomenon. Recent reports from McAfee [17] and Enisa [8] confirm that all malicious activities on the Internet show no disruption of their steady growth, botnets included. Attackers are also widening the spectrum of targeted platforms: In April 2012 it was reported that more than 600,000 Apple computers [7] were infected by the `Flashback` malware, which turned these machines into bots. Moreover, mobile threats have experienced an outstanding increase, becoming a serious problem for Android users, being the mobile OS by Google the most targeted platform.

Also, research by Grier, Ballard, Caballero, Chachra, Dietrich, Levchenko, Mavrommatis, McCoy, Nappa, Pitsillidis, et al. [10] illustrates how the malicious activities are no longer perpetrated for no-profit reasons of miscreants' entertainment, but do involve economical returns. Emergence of the *exploit-as-a-service* business model [10] is a worrying new phenomenon, where exploit kits are sold likewise as a commodity for attackers' needs.

Botnets themselves are becoming infrastructures that are unashamedly crafted for lucrative goals. `Torpig` and `Zeus` for instance aim at stealing financial accounts credentials from the infected machines and send them back to the attackers. Another lucrative kind of malware is *ransomware*. Ransomware is a not new, yet never as scary, threat that has made the news lately with the coming of `Cryptolocker`, a malware able to collect revenues that were *conservatively* estimated at 1,100,000 USD [25]. The communication between the infected machines and the attacker, and the infection itself were realized by the means of a botnet.

Therefore the mitigation of botnets is of primary interest for defenders, and to this end their efforts focus on unveiling the IP addresses of the so called Command & Control Servers (C&C hereafter). A C&C Server is the machine from where the attackers dispatch their orders to and collect data from the infected machines. In a centralized botnet architecture (the most prevalent), once the communication between the C&C and the infected machines, the *bots*, is disrupted, the latter become *dormant* and usually harmless.

Attackers are well aware of the most precious asset and weakness in their infrastructure and have developed increasingly sophisticated techniques to make the disruption of the C&C channel a fatiguing endeavor on the defenders' side. To this end, most of nowadays' centralized botnets' communication relies on the employment of Domain Generation Algorithms (DGAs hereafter), which make the *rendezvous* point (i.e., the moment when the bot *finds* the botmaster) unpredictable and the communication channel resilient to interruptions. These algorithms generate lists of random domains every τ time, say daily, sometimes employing unpredictable seeds, which the bots try to contact. The attacker registers one domain and awaits for the *zombies* to reach the correct URL. When this happens the communication can start and data can be transmitted in both directions.

The mitigation of botnets is a topic widely covered in literature, [2] [4] [6] [18] [22] [24] [26], where many authors propose detection tools which, yet effective, mostly rely either on a *supervised* approach or on clients' IP addresses or on both, two aspects that we deem as shortcomings. In fact, the supervised approach requires labeled data to be fed to the system, which means that before the detection can be actually started, there has to be another way to track the threat and label it as such. Moreover this approach suffers from new menaces that exhibit features previously not considered, as a change in the DGA. Leveraging clients' IP addresses is a shortcoming as it leads to difficulties related to the clients' privacy and the deployment of a monitoring system at lower levels of the DNS hierarchy.

In 2013, Schiavoni et al. [22] proposed Phoenix, a detection system that meets most of our criteria, as it does not require any *a priori* knowledge and it analyzes passive DNS data, free of any privacy issues. Phoenix is able to *discover* clusters of domains used as aliases for botnets' C&C servers, thus unveiling their IP addresses, and to *detect* malicious domains, using the knowledge (i.e., the clusters) previously generated. Despite these valuable features, Phoenix is not able to detect *i)* threats that use C&C servers with previously unseen IP addresses, *ii)* threats that dynamically slightly modify the DGA, e.g., by changing the set of TLDs used and *iii)* it lacks of validation in the wild.

We propose Cerberus, a detection system that overcomes the aforementioned limitations of Phoenix, being able to detect previously unseen threats in the wild using an *unsupervised* approach. Cerberus goes through three stages to achieve its goals. The first phase is called **Bootstrap Phase**, where it uses Phoenix to generate the ground truth, later to be employed in the **Detection Phase**, in an unsupervised and automatic fashion. The ground truth consists of a list of clusters of domains related to DGA-based malicious activities, e.g., botnets and trojans. This ground truth is not necessary to Cerberus to function, as it is able to operate in a complete unsupervised fashion.

After the system has been bootstrapped, it starts analyzing a live stream of DNS passive data, which consists of DNS replies: the domain name, the IP address it resolves to and the TTL. This data needs to be filtered from legitimate domains. To this end during the **Filtering Phase**, we apply a list of heuristics to the data, heuristics that take into account parameters as the registration date, the TLD employed and the TTL. The domains that remain at the end of the filtering process are to be considered likely to be malicious.

These domains are the input to the aforementioned **Detection Phase**, where we try to label this data using the ground truth generated in the **Bootstrap Phase**. To this purpose, Cerberus sees whether the unseen domain d shares its IP address with the clusters: If this is the case we train a Support Vector Machine equipped with the Subsequence String Kernel [16] function and we use it to label d. Otherwise, we start keeping track of the IP of d, l. This means that we record the likely-to-be- malicious domains that resolve to l throughout time. After Δ time of recording, Cerberus groups these suspicious IPs by the Autonomous System that they reside in and performs a clustering routine that leverages the DBSCAN algorithm and the Subsequence String Kernel as a metric. These clusters are then added to the ground truth and the increased knowledge is then used to analyze new live DNS data.

We used one week of real passive DNS data collected by a ISC/SIE passive DNS monitor to test our system. During the first week Cerberus classified 187 malicious domains, 167 of which belonged to the `Conficker` threat, previously discovered by Phoenix. At the end of the week we had collected 1,300 suspicious IP addresses, for a total of 3,576 domains. Then the clustering routine was able to extract 47 new clusters, featuring known malicious IP addresses of threats as `Palevo` and `Sality`. The clusters were added to the previous knowledge and the day after CERBERUS was able to detect 319 malicious domains.

Document Organization

The remainder of this document is organized in the following fashion:

- Chapter 2 provides the background knowledge needed to understand the phenomenon we aim at analyzing. We give a definition of what a botnet is in Section 2.1, why the attackers are interested in Section 2.1.1 in setting up this type of infrastructure, how a botnet can be structured in Section 2.2, the communication system between the botmaster and the bots in Section 2.3, focusing on the Domain Generation Algorithms in Section 2.4 and the countermeasures that can be applied to mitigate this threat in Section 2.5. Then we present two case studies of botnets active in the wild, `Torpig` and `Cryptolocker` in Section 2.6;
- Chapter 3 discusses why we want to study this phenomenon, in Section 3.2 we present the current the state of the art we wish to improve and in Section 3.3 the goals and challenges we mean to achieve and face, respectively;
- Chapter 4 introduces CERBERUS, a detection system for DGA-based malicious activities. In Section 4.1 we give an overview of the system, while in Section 4.2 we discuss in details all of its aspects;

- Chapter 5 explains how Cerberus was implemented and the technologies employed, describing the overall architecture (see Section 5.1)), the process in Section 5.1.1 and then the actors involved in Section 5.1.2 during the system's lifecycle. Then we focus on the most crucial parts and provide a more detailed description of those in Section 5.2;
- in Chapter 6 we test the effectiveness of our system. To this end we run two tests. The first one serves to confirm the accuracy of the SVM Classifier, and it employs the clusters generated by Phoenix as dataset. For the second one instead we use DNS passive data collected by a ISC/SIE DNS monitor. We let Cerberus analyze one week of data and look at the results, which look promising;
- Chapter 7 analyzes the shortcomings Cerberus is affected by and proposes possible remedies to them, along with a few possible enhancements and future developments. Finally, it sums up what we achieved in this work;

CHAPTER 2

BOTNETS

A botnet is a network of compromised machines, remotely controlled by the *botmaster*. These malicious infrastructures are deployed by the attackers for profit, and the more machines are compromised, the more powerful the infrastructure, the more profitable the business. In the last years we have witnessed an aggressive spread of this phenomenon that *defenders* from both the industry and the academia are striving to mitigate. In this chapter we will tackle the different aspects of botnets in order to give a brief yet insightful overview of this threat.

Chapter Organization

The remainder of the chapter is organized in the following fashion:

- in Section 2.1 we explain what is a botnet;
- in Section 2.2 we explain the different topologies a botnet can feature;
- in Section 2.3 we explain how botnet's hosts communicate;
- in Section 2.4 we deeper analyze a certain way to communicate, on which we focus in CERBERUS;
- in Section 2.5 we overview the possible techniques to mitigate a botnet;
- in Section 2.6 we provide two case studies to better explain why this phenomenon is a serious and expanding threat.

2.1 What is a botnet?

A botnet is a network of compromised machines called *bots* or *zombies* under the remote control of a human operator called *botmaster* [9].

The *bot* is the piece of software that infects and compromises the machines. The infection is carried out through a variety of so-called *distribution channels*, which vary from compromised websites that serve malware via drive-by mechanisms to phishing. Once infected, the machine will continue to work as nothing changed to the eyes of the legitimate user, while it is now capable of executing malicious activities on the behalf of the *botmaster*, who will employ a Command and Control Server (C&C) to dispatch orders to and gather information from the *zombies*. What is the nature of these criminal activities are addressed in the next paragraphs.

2.1.1 Botnet Purposes

The main purpose of a malicious botmaster is to recruit as many *zombies* as possible in his army, as any botnet activity augments its effectiveness by the number of *bots* involved. In the following of this section we depict *a few* scenarios of malicious activities carried out employing this type of infrastructure.

Information Gathering

Botnets of this type aim at stealing personal data of various kind from the infected machines. `Torpig` for instance, which we shall further analyze in Section 2.6.1, was crafted with the precise intention of stealing financial accounts credentials and credit or debit card numbers. Beside personal data, we can think of a scenario where the infected machines reside inside a

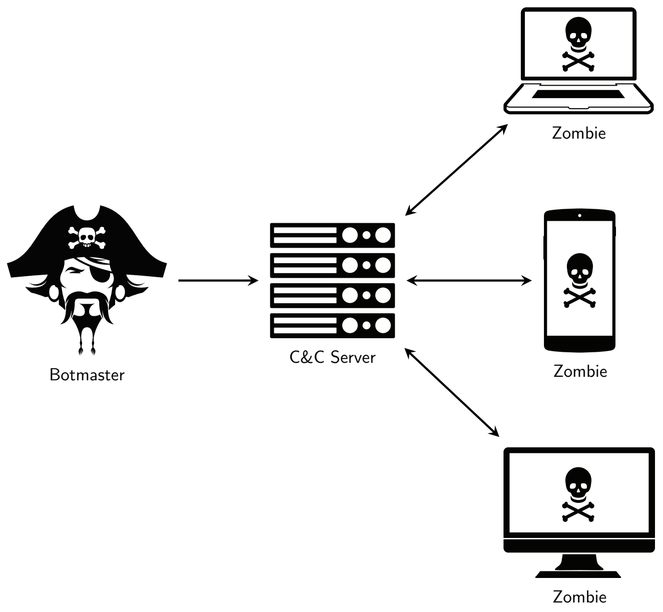


Figure 1: An example of a botnet.

corporate network. In this case the targets would be trade secrets, as intellectual properties or management plans.

Distributed Computing

The network of infected machines can be leveraged to perform computing-intensive tasks. One of the most interesting ones is Bitcoin mining: Cybercriminals aim at creating a botnet of infected machines forced to perform complex calculations to earn them money, putting the machines under heavy CPU and GPU load [25]. `Trojan.Coinbitminer` is an example of this type of malware[1].

[1]`http://www.symantec.com/security_response/writeup.jsp?docid=2011-072002-1302-99`

Spamming

Spamming is one of the most known malicious activities. Spamming botnets are used to send out a high volume of e-mails with various purposes, amongst which we find malware spreading as an e-mail attachment (the `Cryptolocker` malicious binary was spread in this fashion in its early versions, see Section 2.6.2), frauds, XSS attacks and advertisement of malicious services, basically using the bots as mail transfer agents without the victims noticing. It is not easy to set up and keep active such an infrastructure, as spam sources get blacklisted and the messages are no longer delivered. By employing a *distributed* rather than *centralized* message dispatching servers architecture, the attackers aim at and succeed in circumventing this obstacle.

Distribute Denial of Service

Distribute Denial of Service (DDoS) is an attack where a machine receives an excessive amount of requests. To overcome the overwhelming volume of incoming traffic, usually the machine interrupts its services. This results in economic loss, especially when the company business heavily depends on the offered service online availability. In this case the bots are commanded by the botmaster to send an excessive amount of requests to the targeted machine.

Malware Diffusion

Once the *bot* program infects a machine, it can be used to install *further* malware. An example of this behavior comes from the recent Cryptolocker *ransomware*: Its primary vector of infection was the `Zeus Gameover` P2P botnet, beside being spread via e-mail spamming.

Summary

In this section we have defined *what* is a botnet and discussed *why* an attacker would struggle to create this type of networks. In the next section we want to present the different topologies a botnet can be structured into and explain why we chose to target the *centralized* architecture.

2.2 Botnet Topologies

Different botnet topologies, imply different benefits and weaknesses. Our study will focus on the centralized architecture, as it is the most spread, even though recent reports [8] indicate a rise in the P2P topology. Remarkably, certain P2P botnets automatically fall back to centralized topologies in case of failures to avoid starvation of the bots.

2.2.1 Centralized

This topology reflects the classic and well-established *client-server* pattern. The bots communicate directly with the botmaster [22], which forwards messages between clients [4]. This technique guarantees *i)* low latency and *ii)* control over the packet delivery. On the other hand its weaknesses are caused by *i)* a single point of failure, if the C&C is compromised the whole botnet is, and *ii)* are easier to detect, since many clients connect to the same point [4].

2.2.2 P2P

The main advantage in employing a P2P technology consists of a much more robust and resilient infrastructure, as we do not have anymore a single point of failure, but each bot is responsible of broadcasting the message received to the other *zombies*. However, the design of P2P systems are more complex and there are typically no guarantees on message delivery or latency [4].

2.2.3 Unstructured

This is another way to design a botnet, featuring *zombies* that are completely agnostic with respect to the same botnet they belong to. Whenever they need to send a message to the infected network, they encrypt it, randomly scan the Internet and pass along the message when they detect another bot. Even though the design is quite simple, it would not be able to guarantee the actual deliver and it would also be prone to extremely high latencies.

Summary

We have presented three different topologies a botnet can be structured into and explained that we chose the *centralized* architecture as it is the most used by the attackers. Now we want to understand how the *zombies* communicate with the *botmaster*.

2.3 Communication Systems of DNS-based Botnets

Every distributed network of machines must be equipped with a communication protocol, as communication intra hosts is a defining feature of these architectures. Before starting the exchange of malicious commands, the *bots* and the *botmaster* must establish a connection. In centralized botnets, this happens through the C&C Server in the so called *rallying phase*, when the bots and the C&C Server find a *rendezvous* point.

In the following of this section, we aim at describing the *rallying* phase in detail, presenting three different ways through which the *rendezvous* can be found. One is the evolution of the previous one, evolution dictated by the necessity of finding a more resilient and robust technique, and they all refer to the *centralized* topology. Before that we would like to summarize a few preliminary concepts needed in the following of this chapter.

2.3.1 Preliminary Concepts

We deal with two key concepts that need to be understood before we start dealing with botnets: The *domain names* and the *DNS* technology and infrastructure.

Domain Names

Domain names are a way to help humans remembering resources' locations on the Internet, otherwise only traceable by their IP address. A domain name is a sequence of *labels*, separated by dots. The labels are to be read from right to left as to respect the hierarchy they reflect. Take for instance the domain `www.example.com`. This domain name is composed by three labels or *subdomains*. Starting from the rightmost one, we find the `.com` label. This is called *Top Level Domain* and it comprises the generic top-level domains (gTLDs) and the country code top-level domains (ccTLDs).

$$\overbrace{\texttt{www}}^{\text{third level domain}} \;.\; \underbrace{\texttt{example}}_{\text{second level domain}} \;.\; \overbrace{\texttt{com}}^{\text{top level domain}}$$

Figure 2: example.com labels hierarchy.

The next label in the example is `example` and it is called *second level* domain. Along with the TLD it forms a *hostname*, which is a domain name to which corresponds an IP address, i.e., an actual *machine*. We say that `example` is a subdomain of `com`. Next on we find `www`, which identifies a subdomain of `example.com` (see Figure 2).

The labels hierarchy can count up to 127 levels and a domain name cannot be longer than 253 ASCII characters, though practical implementations may impose more strict limits. How an IP address is retrieved when using a domain name is achieved by the Domain Name System (DNS) and we explain it in the following.

The Domain Name System

The DNS is a hierarchical and distributed infrastructure of databases and servers that provide the translation from domain names to IP addresses.

It is *hierarchical* because no single database contains the, for instance, the whole `www.example.com` entry, but we have a distinct node for at least each one of the first three levels of hierarchy: The first one is the **root server**. The root server looks at the first label of the domain, the TLD, and redirects the DNS query to the right **TLD server**. The TLD servers then look at the second level domain, `example`, and redirect the query to the correct **authoritative server**. Authoritative servers are under the authority of the organizations that register them, and contain (at least) the domain-name-to-IP record.

The actual querying procedure requires another actor, the **local DNS server** (see Figure 3). Local DNS servers are used to reduce the load of servers located at higher levels of the hierarchy by caching the results. In fact, when an authoritative server replies, it sends also a `Time To Live` parameter, which sets for how long a local DNS server can reply using the cached value rather than asking the root server again.

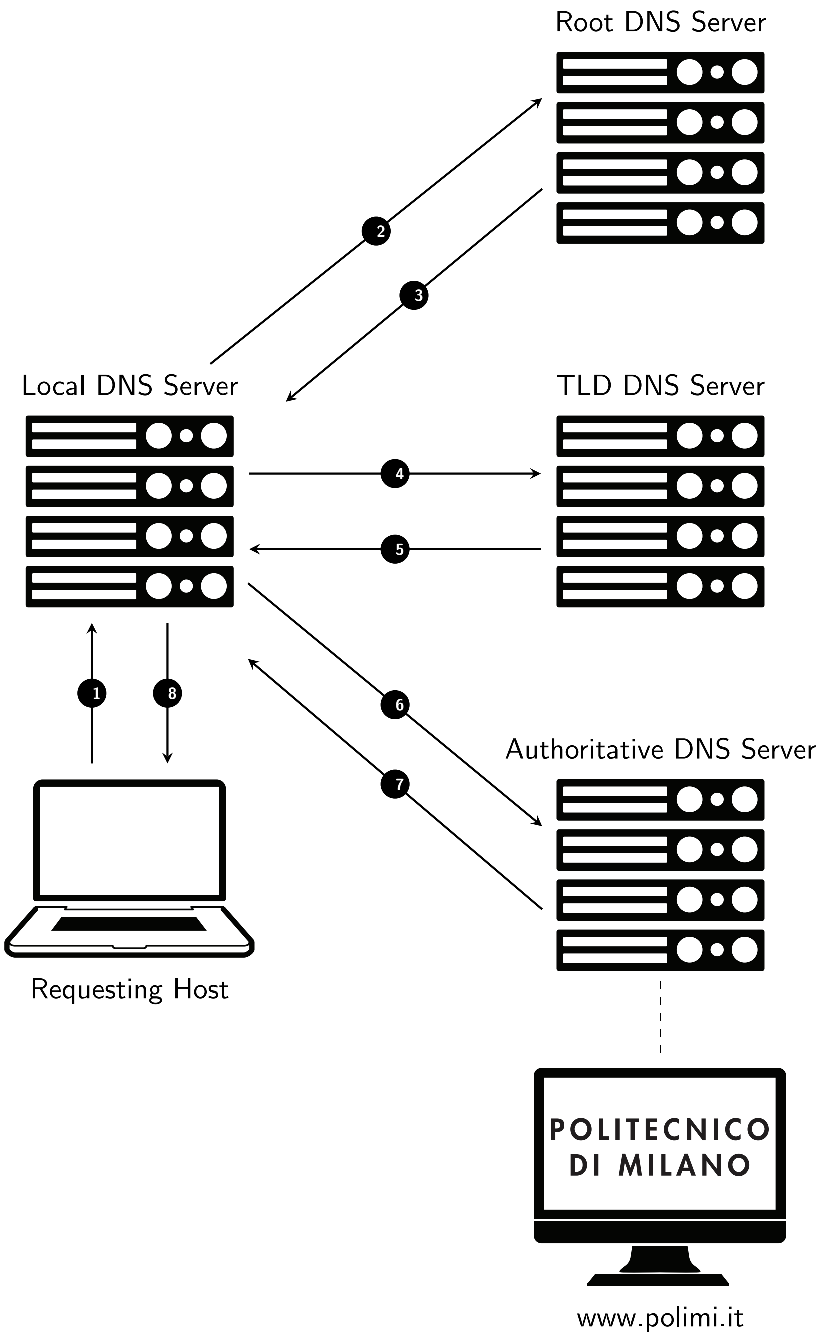


Figure 3: DNS resolution of `www.polimi.it`.

2.3.2 Command & Control Channel

The Command & Control Channel (C&C) is the logical communication channel between the *bots* and the C&C Server, i.e., the *botmaster*. It is a bidirectional channel that allows the *botmaster* to dispatch orders to the *bots*, and the *bots* to send the harvested data or feedback back to the *botmaster*, depending on the purpose and functioning of the botnet.

A C&C Channel is established after the malware connects the zombie to the C&C server. Upon establishment of the C&C channel, the *zombie* becomes part of attacker's botnet army [9]. Therefore, if the defenders managed to tear down the C&C channel, *i)* the botmaster would not be able to send orders to the *bots*, and *ii)* the zombies would not be able of sending the data they have collected, making the infrastructure harmless.

Single Point of Failure

It is no surprise that defenders concentrate their efforts in blocking C&C communications as a medium to disable botnets [22]. To this end, the main techniques used are **takeover**, where you take control of a botnet, and **sinkholing**, where you just interrupt the C&C Channel. We will focus on these techniques in Section 2.5, what is important now is to understand that if an attacker wants to keep the infrastructure alive, he has to build a robust and resilient C&C Channel. In the following of this section we present three different techniques to address this matter.

2.3.3 Rallying Techniques

A *rallying technique* is a protocol to establish a connection, i.e., the C&C Channel, between the *botmaster* and the *bots*. In a centralized topology, it is bots' responsibility to find the

C&C Server, as the latter is agnostic with respect to the location of its army. In the next paragraphs we will describe three techniques, in ascending order of complexity and resiliency to the defenders' countermeasures.

Hardcoded IP Address

The simplest way to instruct a bot to connect to a server is to *hardcode* the endpoint's IP address in the program itself (see Figure 4), or to ship the malware with an external file containing the IP address to contact. There are a few improvements that can be adopted, as providing a *list* of *rendezvous* IPs rather than a single one, and update the list throughout time, in case the *botmaster* is forced to migrate the C&C Server somewhere else.

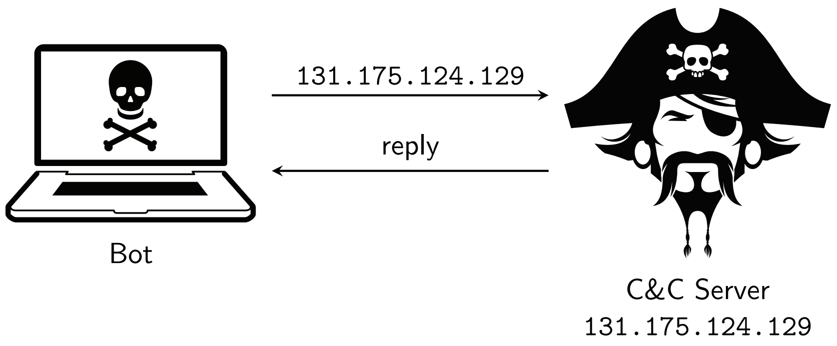


Figure 4: Harcoded IP.

Nevertheless this technique is vulnerable to *information leakage*: Once the defenders get the malware binary and are able to reverse-engineer it, they know the *rendezvous* points. Once this information is acquired, they can set up a strategy to simultaneously *sinkhole* all the C&C servers and make the infrastructure harmless.

Hardcoded Domain Name

A first improvement consists in leveraging the DNS protocol, and ship the malware with a (list of) domain to query to obtain the IP address of the C&C Server [22]. In this way the attacker is free to move the C&C Server to different locations (i.e., different IP addresses) and update the DNS records to let the domain point to the new IP address. This leads to a communication protocol that is more robust and resilient than hardcoded IP addresses. Defenders in this case have to intervene at the DNS level and seek for registar authorities' cooperation in sinkholing the domain names. A successful case of this kind of operation is the takeover of a `Zeus` botnet by researchers at Trend Micro [21].

Although the countermeasures in this case are harder to be played out, this technique still suffers from the *leakage* problem: Once the defenders get their hands on the malware binary and are able to reverse-engineer it, they know the rendezvous points.

Domain Generation Algorithms

A Domain Generation Algorithm (DGA hereafter) is an algorithm used to automatically generate domain names, so called Automatically Generated Domains (AGDs hereafter). Most of modern botnets leverage this technique in the rallying phase: Both the *botmaster* and the *bots* produce a list of pseudo-random domains, one of which is registered and functions as *rendezvous* point. As our work heavily relies on the malicious AGDs detection, we further analyze this technique in Section 2.4.

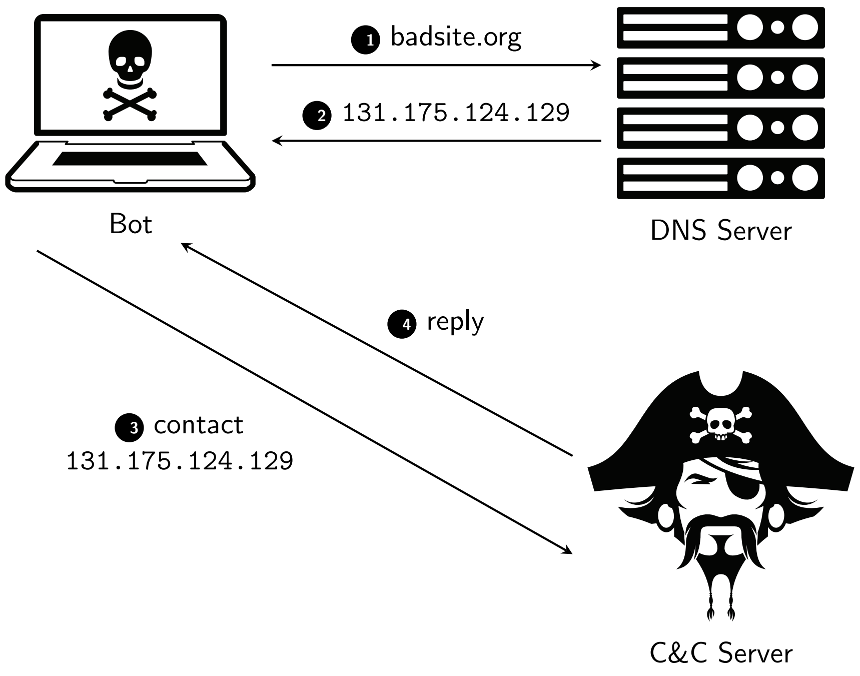


Figure 5: Harcoded domain.

Summary

In this section we have presented *how* in a botnet the communication between the *bots* and the *botmaster* is achieved. We have analyzed the evolution of the *rallying techniques* that attackers have implemented throughout time to make their network more resilient to the defenders' countermeasures. The primary and most resilient technique employed nowadays consists of the generation of random domains, as briefly described in Section 2.3.3. In the next section we provide a deeper analysis of this protocol, as CERBERUS focus its detection technique on it.

2.4 Domain Generation Algorithms

In the previous section we have briefly mentioned the use of DGAs as a rallying mechanism to set up the communication channel in a botnet. In this section we analyze in detail the features and the weaknesses of such a technique.

2.4.1 The Idea

Hardcoded IP addresses and domains suffer from *information leakages*. In DGA-based botnets the malware that runs each bot is equipped with *instructions* to generate possible *rendezvous* locations, rather than with the locations themselves. A DGA uses one or more random seeds to initialize the generation of N random strings, where N can vary up to several thousands of domain produced at each routine.

Both the *botmaster* and the *bots* produce the same list of random domain names at the same time. The botmaster then contacts a registar authority to register just one or a few of them. Unfortunately, there are registrars that do not enforce strict checks on the final ends of the registered domains, and simply allow whoever pays to register whichever domain name[1]. When the rallying phase starts, the *zombies* begin to try to contact the domains in the list. If the domain is not registered, i.e., the DNS server replies with an `NXDOMAIN` answer, the zombie continues to query the domains. As soon as the DNS server delivers a valid answer, the bot contacts that domain and the C&C Channel is established (see Figure 6).

[1] `http://krebsonsecurity.com/wp-content/uploads/2012/03/rogue_registrars_2012_DRAFT.pdf`

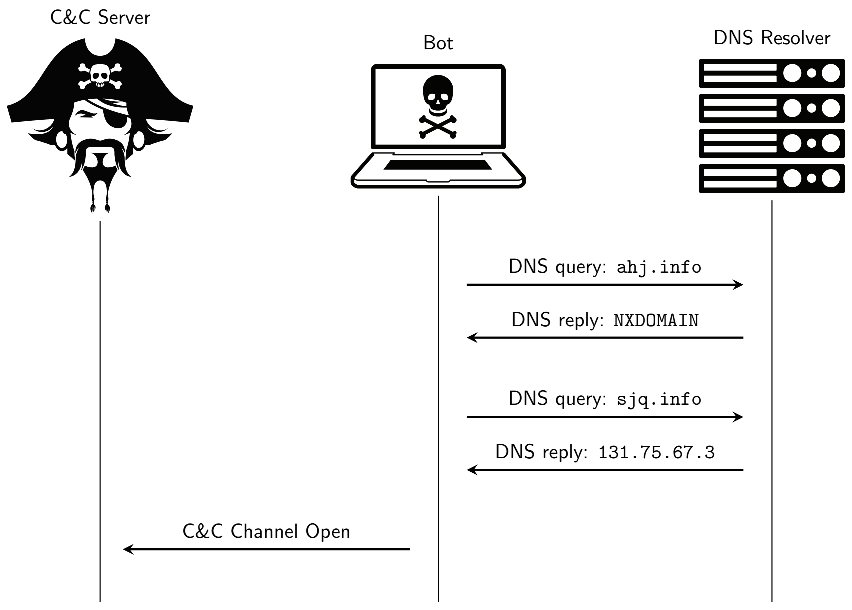


Figure 6: DGA rallying mechanism.

2.4.2 The Choice of the Seed

The first botnets to adopt this rallying technique had simple DGAs. For instance the `Torpig` DGA relied on the date and on a hardcoded constant to generate the list of domains. Moreover this list would count up to six possible domains a day, three of which valid throughout the whole week. This naïve implementation lead the defenders to takeover the botnet by registering in advance a list of all the possible domains to be contacted in the close future [26].

In order to get around this issue, attackers started producing a much higher volume of domains: `Conficker.C` produces batches that count 50,000 daily domains. This entails a strong *asymmetry* between the endeavors that the defenders should cope with to register all the possible domains and the effort on the miscreants' side, who have to register just *one* domain. The economics of such a brute force approach make it impossible to be played out.

Another technique involves the choice of an *unpredictable* seed, for instance the current Twitter trending topic, making impossible every precautionary registrations of domains as neither the botmaster nor the defenders can know what this seed will be at the moment of the domains' creation.

2.4.3 Migration Strategy

By using DGAs, attackers can set up a very robust migration strategy. It is almost no use for the defenders to retrieve the list of random domain names. Once a domain is used to set up the C&C, it is no longer valid: It is then useless to blacklist it. The only way to disrupt the communications is to retrieve the IP address of the C&C Server. Still, once this information is acquired, it suffices to change the IP address the new random domains will resolve to, to escape the defenders' handcuffs. The next time the bots will try to establish the C&C Channel they will use new domains and new IP addresses, thus making the creation of the communication untraceable.

2.4.4 Side Effects and Weaknesses

DGAs have at least two side effects. We report them as they are used as a detection strategy in many works in literature. Moreover we will exploit the second one in our own detection system.

NXDOMAINS

The use of DGAs causes high peaks of DNS requests that return an `NXDOMAIN` answer. This phenomenon can be traced by analyzing the DNS traffic, and hosts that exhibit this behavior are likely to be infected. This approach was successfully followed by Antonakakis, Perdisci, Nadji, Vasiloglou, Abu-Nimeh, Lee, and Dagon [3] and, when it is easy to have access to the clients' IP address, it remains the most effective technique to track down DGA-based malware, even though it involves privacy related issues due to the required access to the infected machines' IP addresses.

Random Names

It is quite easy for a person to distinguish between an AGD and a domain created by a human. For instance `zz87ihfda88.com` quickly calls attention, while a domain as `facebook.com` is readily recognized as "normal". Schiavoni et al. [22] leveraged this feature to automatically distinguish the ones from the others.

2.5 Botnet Countermeasures

In this section we present two countermeasures to the botnet phenomenon. The first one, *sinkholing*, is a *passive* technique that aims at interrupting the communication between the bots and the botmaster: The defenders do not care about anything else. The second one, *takeover*, is an *active* technique where the defenders rather *hijack* than disrupt the communications, thus acquiring control over the infrastructure.

2.5.1 Sinkholing

Sinkholing is a countermeasure to mitigate botnets that target the C&C Channel. The defenders stop the communications between the bots and the botmaster (see Figure 7). This is achieved usually by leveraging the DNS infrastructure once the IP of the C&C Server is known. Defenders can modify the DNS answers returned by DNS servers as to point to machines under their control. In this scenario they are not interested in controlling the botnet, or accessing the information harvested by the bots, but only in blocking the communication, thus making the threat harmless.

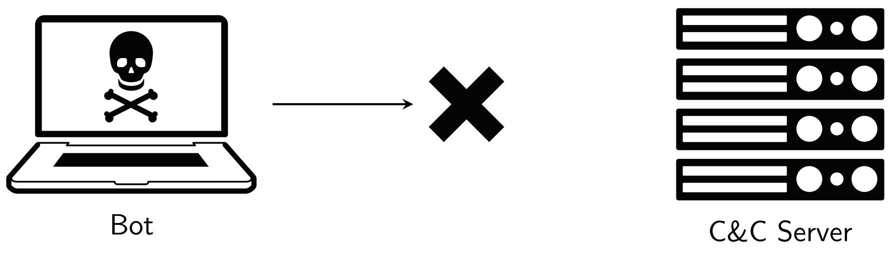


Figure 7: Sinkholing.

2.5.2 Takeover

When defenders succeed in a *takeover* operation it means that they successfully took control over a botnet (see Figure 8). This operation requires two stages.

The first one can be seen as a *sinkholing* operation, where they have to ensure that the bots communicate with machines under their controls rather than with the C&C Server. This can be achieved in at least two ways. Stone-Gross et al. [26] managed to reverse engineer the `Torpig` DGA, then generated the list of all the possible AGDs in the close future and registered them. Sancho and Link [21] instead contacted the registar authority and asked them to collaborate by rerouting the traffic directed toward a domain that identified a `Zeus` C&C.

In the second stage the machine(s) under the defenders' control must mimic the behaviour of the C&C server previously in charge. If they succeed in the mimicking, the bots would start sending the harvesting data to and obey to orders received from the defenders. We find in literature [26] and in industry reports [21] successful examples of takeovers, which greatly helped to shed some light on *how* a botnet works. In Section 2.6.1 we shall further analyze what Stone-Gross et al. [26] managed to achieve.

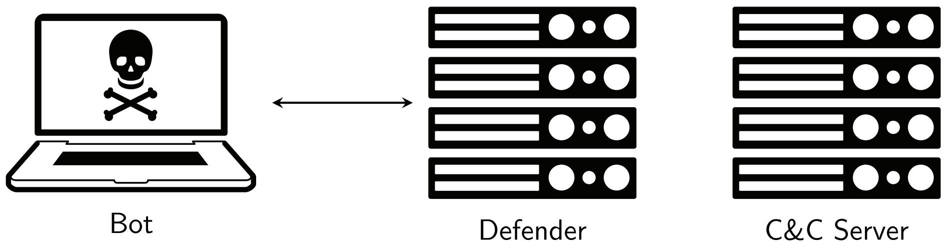


Figure 8: Takeover.

2.6 Botnets: a Modern Threat

In the previous sections we have described the botnet phenomenon, highlighting the technologies employed to build this kind of infrastructure and the countermeasures used by the defenders to mitigate this threat. In this section we give some quantitative data to highlight the severity of the menace and explain why defenders care about it and are struggling to stem its diffusion.

In Section 2.6.1 we propose a summary of [26], where the authors managed to take control of the `Torpig`, one of the first botnets, for ten days, and managed to see and quantify what this kind of infrastructure is capable of.

In Section 2.6.2 we analyze `Cryptolocker`, a type of malware that falls into the category of *ransomware*. `Cryptolocker` appeared at the beginning of September 2013, and was responsible of more than 250,000 infected machines, leading to earnings that sum up to the order of magnitude of millions of USD.

2.6.1 Torpig

When Stone-Gross et al. [26] started to track `Torpig`, they managed to take control of the whole infrastructure for a period of ten days time. This was the first ever reported case of takeover, which unveiled unique details about techniques and modus operandi behind botnets. Over the years, subsequent researches inspired by [26] adopted similar and novel methods to track and counteract botnets.

DGA

They managed to fully reverse the DGA, which is reported in Listing 2.1. `Torpig` DGA is capable of producing up to six domain names every day. First it computes a *weekly random string*, which remains the same for seven days. After appending `.com`, `.net` and `.biz` to the random string the bots try to contact the resulting domains. If it is not able of establishing a communication with any of them, it then computes a *daily random string*. Same as with the *weekly* case, the three TLDs are appended to the random string and the corresponding domains are contacted.

Stone-Gross et al. [26] noticed a few interesting facts:

- the Domain Generation Algorithm is seeded with a *hardcoded* numerical parameter and the current date, no unpredictable seed is used;
- there are only six domains generated every day, three of which stay the same for seven days.

Takeover Strategy

These observations gave the researchers a competitive advantage, which lead them to pre-emptively register all the possible domains for three consecutive weeks, from January 25th to February 15th, 2009 [26] and set up a copycat of the infrastructure employed by the attacker to collect the data. The takeover successfully started on the 25th and ended prematurely on February 4th, due to a change in the DGA algorithm.

Data Collection

Torpig is a malware crafted with the precise intention of stealing financial data. In ten days Stone-Gross et al. [26] were able to collect credentials of 8,310 accounts at 410 financial institutions and 1,660 unique credit and debit card numbers.

Revenues

Even though a precise estimation is far from trivial, the authors indicate that in ten days activity the Torpig controllers may have profited anywhere between 83,000 and 83,000,000 USD [26].

Listing 2.1: Torpig DGA Python implementation Stone-Gross et al. [26].

```
suffix = ["anj", "ebf", "arm", "pra", "aym", "unj", "ulj",
          "uag", "esp", "kot", "onv", "edc"]

def generate_daily_domain():
    t = GetLocalTime()
    p = 8
    return generate_domain(t, p)

def scramble_date(t, p):
    return (((t.month ^ t.day) + t.day) * p) + t.day + t.year

def generate_domain(t, p):
    if t.year < 2007:
        t.year = 2007

    s = scramble_date(t, p)
    c1 = (((t.year >> 2) & 0x3fc0) + s) % 25 + 'a'
    c2 = (t.month + s) % 10 + 'a'
    c3 = ((t.year & 0xff) + s) % 25 + 'a'

    if t.day * 2 < '0' or t.day * 2 > '9':
        c4 = (t.day * 2) % 25 + 'a'
    else:
        c4 = t.day % 10 + '1'

    return c1 + 'h' + c2 + c3 + 'x' + c4 + suffix[t.month - 1]
```

2.6.2 Cryptolocker

`Cryptolocker` is a trojan classified as *ransomware*, because it asks the victim for a ransom to get back the documents of hers that the malware has encrypted right after infecting the machine. Ransomware malware has become an increasing problem [17] in the most recent years. `Cryptolocker` has appeared at the beginning of September 2013 and targets several Windows Systems, including Windows 7. Although Symantec classifies this kind of threat *easy* *to remove*[1], its effects are, in most cases, devastating. In the next paragraphs we analyze what has gained the status of *menace of the year*[2].

Infection Vector

The very first samples seem to have been released on September 5, 2013 [15]. Compromised websites were responsible of the download, i.e., it was a *drive-by-download* attack, where you just need to visit or "drive by" a web page, without stopping to click or accept any software, and the malicious code can download in the background to your device[3]. Then the attackers chose to change the infection vector, shifting from compromising websites to sending spam e-mails. Business professionals were the targets of this second phase: Even the Swansea Police Department, Massachusetts USA, had to pay 750 USD to recover images and documents that had been encrypted by `Cryptolockker`[4]. They would receive fake "consumer complaints" against either themselves or the organization they belong to. Attached to these e-mails was a `zip` archive with a random alphabetical filename containing 13 to 17 characters [15]. Once decompressed, you would find in the archive a single file inside, an executable actually, featuring the `exe` extension for executables in the Windows environments.

[1]`http://www.symantec.com/security_response/writeup.jsp?docid=2013-091122-3112-99`

[2]`http://www.symantec.com/connect/blogs/cryptolocker-qa-menace-year`

[3]`https://blogs.mcafee.com/consumer/drive-by-download`

[4]`http://www.theguardian.com/technology/2013/nov/21/us-police-force-pay-bitcoin-ransom-in-cryptolo`

On October 7, 2013, researchers from the Dell SecureWorks Counter Threat Unit observed yet another shift in `Cryptolocker` distribution. `GameOver Zeus`[1] would now be responsible for the *ransomware* spreading and installation. The user would receive a spam e-mail, featuring a fake Adobe Reader icon attachment. Once opened, this would lead to the download of the `Upatre` malware: It would execute `Gameover Zeus`, which finally would download and install `Cryptolocker`. According to [15] at the time of their publication, December 18 2013, this last infection vector is the primary.

```
Dear Mr. Doe,
Please find the attached copy invoice which is showing
as unpaid in our ledger.

I would be grateful if you could look into this matter
and advise on an expected payment date.

Many Thanks
John Smith
```

Figure 9: An example of a Cryptolocker spam e-mail.

[1]`http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_Peer_to_Peer_Gameover_ZeuS/`

DGA

Cryptolocker now leverages a Domain Generation Algorithm to produce a list of URLs. This algorithm uses the results of calls to Windows APIs (day, month and year) as initial seed, and is capable of 1,000 different pseudo-random domains at once. Let us go through the several stages [1].

The Initial Seed Cryptolocker gets its initial seed, the current date, either from two Windows APIs: `QueryPerformanceCounter` or `GetTickCount`. If the first API call is successful, the result is stored in the `ECX` register, otherwise the second one is called.

Generating The Array of Seeds After the initial seed is obtained, Cryptolocker generates an array of 623 further seeds combining `SHR`, `XOR`, `IMUL` and `ADD` operations.

Computing the Key During this stage another set of operations are performed, which we will not report here because of none interest, that produce a new key as result, to be employed in the DGA generation step.

Domain Generation Algorithm You can find in Listing 2.2 the Domain Generation Algorithm pseudocode. As you can see starting from the key computed in the above steps (`KEY`) and computing three new keys employing the current day, month and year, it is able to generate the random domain name (`ServerName`).

[1] `http://blog.fortinet.com/A-Closer-Look-at-Cryptolocker-s-DGA/`

TLD The last step consists of attaching the TLD to the random string. Cryptolocker uses seven different TLDs: `.ru`, `.org`, `.co.uk`, `.info`, `.com`, `.net` and `.biz`. Cryptolocker simply iterates this list in order and attaches the TLD.

Listing 2.2: Cryptolocker DGA pseudocode.

```
NewKey = (((KEY * 0x10624DD3) >> 6) * 0xFFFFFC18) + KEY

DayKey = (CurrentDay << 0x10) ^ CurrentDay
if (DayKey <= 1)
    DayKey = CurrentDay << 0x18

MonthKey = (CurrentMonth << 0x10) ^ CurrentMonth
if (MonthKey <= 7) {
    MonthKey = CurrentMonth << 0x18
    if (MonthKey <= 7)
        MonthKey = !(MonthKey)
}

YearKey = ((CurrentYear + NewKey) << 0x10) ^ (CurrentYear + NewKey)
if (YearKey <= 0xF)
    YearKey = ((CurrentYear + NewKey) << 0x18)
}

StringLength = (((DayKey ^ ((YearKey ^ 8 * YearKey ^ ((DayKey ^
    ((MonthKey ^ 4 * MonthKey) >> 6)) >> 8 )) >> 5 )) >> 6) & 3) + 0xC
```

```

do {
    MonthKey = ((MonthKey ^ 4 * MonthKey) >> 0x19) ^ 0x10 *
                                                (MonthKey & 0xFFFFFFF8)
    DayKey = (DayKey >> 0x13) ^ ((DayKey >> 6) ^ (DayKey << 0xC))
                                             & 0x1FFF ^ (DayKey << 0xC)
    YearKey = ((YearKey ^ 8 * YearKey) >> 0xB) ^
                                       ((YearKey & 0xFFFFFFF0) << 0x11)
    index += 1
    ServerName[index - 1] = (DayKey ^ MonthKey ^ YearKey) % 0x19 + 'a'
} while (index < StringLength)
```

C&C Communication

Once the list of AGDs is computed, Cryptolocker tries to contact the C&C Server via HTTP. When it receives a non-NXDOMAIN reply from the DNS server, it encrypts a phone-home message with an RSA public key embedded in the malware itself. Only servers with the corresponding RSA private key can decrypt this message and successfully communicate with an infected system [15].

Encryption

One of them corresponds to the C&C, which will generate a 2048-bit RSA key pair and will transmit the public one to the victim. It is interesting to know that the attackers, rather than implementing their own encryption scheme, leverage Microsoft's CrytpoAPI to build a robust program that is difficult to circumvent [15]. The RSA public key will be used to encrypt files featuring a set of extensions that are likely to indicate personal documents (see Table I).

It is interesting to notice that the attackers chose to target, amongst the others, files with extensions that come from the Adobe suite of professional tools for graphics (Adobe Illustrator, `.ai`), publishing (Adobe InDesign, `.indd`), photo editing (Adobe Photoshop, `.psd`), or Auto-Cad (`.dwg`), a software by AutoDesk used to produce professional CADs, mostly employed by architects. These types of files are usually the results of many hours of work, therefore it is very likely that a user affected by Cryptolocker will be willing to pay rather than starting from scratch. We think this is interesting under a psychological point of view: Rather than personal data, the attackers target the results of laborious work.

TABLE I: FILE EXTENSIONS SEARCHED BY CRYPTOLOCKER.

odt	ods	odp	odm	odc	odb	doc	docx	docm	wps	xls
xlsx	xlsm	xlsb	xlk	ppt	pptx	pptm	mdb	accdb	pst	dwg
dxf	dxg	wpd	rtf	wb2	mdf	dbf	psd	pdd	pdf	eps
ai	indd	cdr	jpg	jpe	jpg	dng	3fr	arw	srf	sr2
crw	cr2	dcr	kdc	erf	mef	mrw	nef	nrw	orf	raf
raw	bay	rwl	rw2	r3d	pef	srw	x3f	der	cer	crt
pem	pfx	p12	p7b	p7c	ptx					

Ransom

Once the documents are found and encrypted, a ransom message will be shown to the user (see Figure 10), where he is informed that some of his files were encrypted, and in order to gain access to them he has to pay a ransom. The ransom varied in the early versions (Table II), but then raised to and remained at 300 USD.

TABLE II: RANSOM AMOUNTS IN EARLY CRYPTOLOCKER VERSIONS.

Amount	Currency
2,000	Czech Koruna (CZK)
1,500	Norwegian Krone (NOK)
1,500	Swedish Krona (SEK)
1,000	Mexican Peso (MXN)
1,000	Danish Krone (DKK)
500	Polish Zloty (PLN)
200	Brazilian Real (BRL)
200	New Zealand Dollar (NZD)
200	Romanian Leu (RON)
100	U.S. Dollar (USD)
100	Euro (EUR)
100	Australian Dollar (AUD)
100	Canadian Dollar (CAD)
100	British Pound Sterling (GBP)

The attackers would let choose the victims a payment method amongst the following ones:

cashU is a safe payment method designed for and customized to suit, serve & support online shoppers in all Arabic speaking and surrounding countries with secure, accessible and easy to use payment solutions, giving everyone the possibility to buy online without discriminating on income, nationality or banking contacts.[1]

Ukash Ukash is an electronic money system regulated by the Financial Conduct Authority that allows users to exchange their cash for a secure code. The code is then used to make payments online, to load cards or e-wallets or for money transfer.[2]

Paysafecard an electronic payment method for predominantly online shopping and is based on a pre-pay system. Paying with paysafecard does not require sharing sensitive bank account or credit card details[3].

Bitcoin Bitcoin is a decentralized monetary system that aims to become the digital equivalent of cash. Like cash, Bitcoin transactions do not explicitly identify the payer nor the payee: a transaction is just a cryptographically signed message that embodies a transfer of funds from one public key to another [25].

Green Dot MoneyPak is yet another pre-paid service to send money. It can be purchased at thousands of stores nationwide (U.S., Ed.), including major retailers such as Walmart, Walgreens, CVS/pharmacy, Rite Aid, Kmart and Kroger[4].

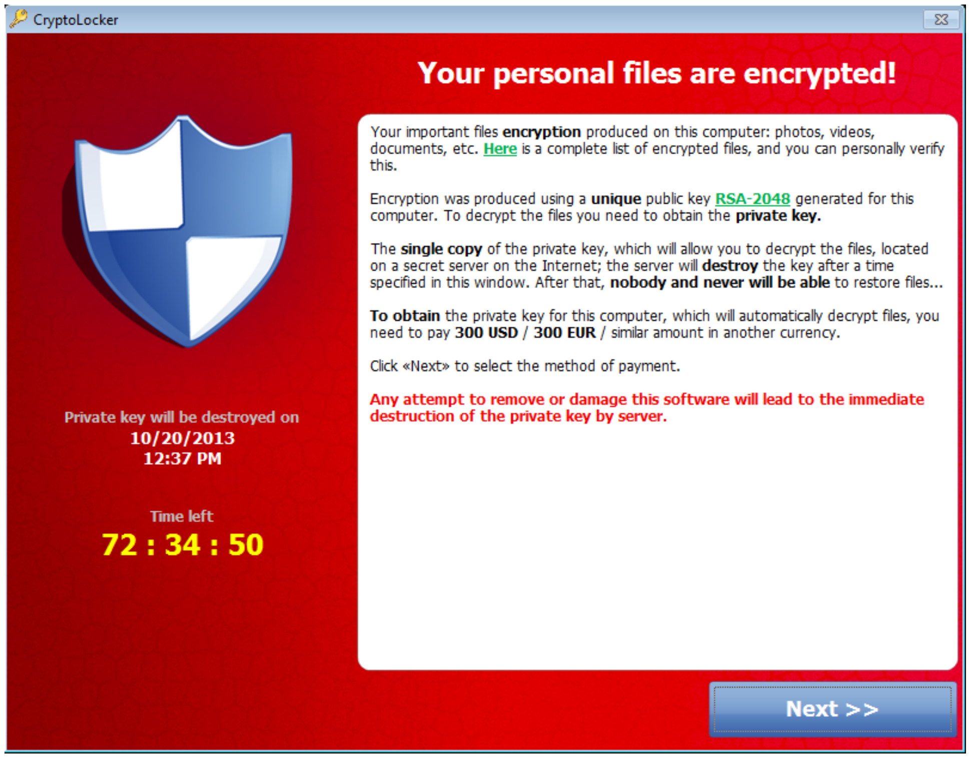
---

[1]`https://www.cashu.com/site/en/aboutCashu`

[2]`https://en.wikipedia.org/wiki/Ukash`

[3]`https://en.wikipedia.org/wiki/Paysafecard`

[4]`https://www.moneypak.com/AboutMoneyPak.aspx`



Your personal files are encrypted!

Your important files **encryption** produced on this computer: photos, videos, documents, etc. Here is a complete list of encrypted files, and you can personally verify this.

Encryption was produced using a **unique** public key RSA-2048 generated for this computer. To decrypt the files you need to obtain the **private key.**

The **single copy** of the private key, which will allow you to decrypt the files, located on a secret server on the Internet; the server will **destroy** the key after a time specified in this window. After that, **nobody and never will be able** to restore files...

To obtain the private key for this computer, which will automatically decrypt files, you need to pay **300 USD** / **300 EUR** / similar amount in another currency.

Click «Next» to select the method of payment.

Any attempt to remove or damage this software will lead to the immediate destruction of the private key by server.

Private key will be destroyed on
10/20/2013
12:37 PM

Time left
72 : 34 : 50

Figure 10: Cryptolocker ransom message.

The Revenues

It is hard to provide a precise estimate of the earnings that Cryptolocker brought to the attackers. Spagnuolo [1] made a rough estimate of Cryptolocker earnings coming from the Bitcoin payment system. In total they identified 771 ransoms, for 1,226 BTC (approximately USD 1,100,000 on December 15, 2013) [25]. Such a measurement was addressed as “conservative”. It is quite clear that the Cryptolocker malware is to be considered of a very lucrative threat.

Conclusions

Cryptolocker is a very interesting example of malware for a few reasons listed here below.

- The primary infection vector is via a botnet: The fact that the attackers chose this vector lead us to think that in the future most of malware will be distributed in such a fashion.
- Cryptolocker forms itself a botnet: The network of infected machines communicate with the C&C server in a centralized fashion, employing a DGA based rallying scheme.
- Being a *ransomware* the malware of the year witnesses the attackers’ paradigm shift, from no-profit malicious activities carried out for hacking interests to profitable ones, capable of massive earnings. “Some men just want to watch the world burn” is no longer an appropriate motto[2] to address the attackers.

[1]Michele Spagnuolo is one of the most brilliant persons I have ever had the chance to know and a friend of mine. Therefore it is a great pleasure of mine to cite his work.

[2]The Dark Knight, 2008 Warner Bros.

- Attackers do not *steal* sensitive personal data, but *lock* computer files that, give the majority of targeted extensions, are likely to be the result of professionals' work, demonstrating a particular attention toward increasing the victims' willingness to pay.

All of the aforementioned reasons fully justify the focus of our work, as modern miscreants' threats leverage the topology and communication scheme we have tailored our software to search for.

2.7 Summary

In this section we have introduced the phenomenon of botnets, one of today's most spread and dangerous threats in computer security. We have briefly overviewed some of the purposes carried out by these malicious infrastructures. We have explained why most of these infrastructures feature a *centralized* architecture and employ a *rallying* mechanism based on Domain Generation Algorithms to establish the C&C Channel, the communication medium used to *i)* send orders to the bots and *ii)* retrieve harvested data or feedback from the bots. Then we focused on the possible countermeasures to mitigate the threat: Both sinkholing and takeover do need the C&C server IP address to be played out. Finally we have reported two real life cases of botnets in the wild, demonstrating the threat danger and the actuality. In the next chapter we focus on the problem we aim at contributing to resolve, i.e., how to track down a botnet.

CHAPTER 3

TRACKING DOWN A BOTNET

Chapter 2 gave an overview of the phenomenon of botnets, which has become one of the most spread and remunerative malicious activities on the Internet. Therefore it is of great interest for defenders to invest time and labor in finding new ways to mitigate them. Given the primary architecture employed (centralized), the most effective way to make a botnet harmless is to track down the C&C Server and interrupt the communication between the *botmaster* and the *bots*.

In this chapter we define the problem of interrupting a botnet activities, present the *state of the art* and state the goals of our work and the challenges we need to solve.

Chapter Organization

The remainder of this chapter is organized in the following fashion:

- in Section 3.1 we precisely define the problem we want to address in this work;
- in Setion 3.2 we present the most recent works in literature that cover this matter, highlighting the points of strength and the shortcomings, both starting point for CERBERUS;
- in Section 3.3 we elicit the goals we want to achieve with CERBERUS and the challenges that have to be faced, highlighted in Setion 3.2.

3.1 Problem Statement

Tracking down and mitigating a botnet is the multi-faceted problem that we wish to address. In the previous section we had an overview on the various topologies, and highlighted how, though P2P botnets are growing, the centralized architecture is still the most popular. Moreover we have explained how most of them employ a *rallying* mechanism based on DGAs.

In Section 2.5 we have seen how the two most effective countermeasures, **sinkholing** and **takeover** both aim at disrupting the C&C Communication Channel. To perform this task it is necessary to know the IP address of the C&C Server.
Therefore the problem of mitigating a centralized DGA-based botnet can be "reduced" to the task of finding the IP addresses of the C&C servers that operate the malicious infrastructure, though this operation is far from trivial.

In the next section we analyze the current state of the art. Throughout this analysis we highlight the major shortcomings of the current solutions and underline the importance of overcoming these limitations and how Cerberus aims at achieving this goal.

3.2 State of the Art

Botnet detection and, more generally, malicious activities detection, by analyzing network data is a topic broadly covered in literature. We focus on the detection of botnets that use DGAs to establish the communication channel and on approaches that analyze high volumes of DNS data.

Works are presented in chronological order: Section 3.2.1 introduces Kopis by Antonakakis et al. [2], which leverages three groups of features to distinguish malicious domains from benign domains. Then, in Section 3.2.2, we report EXPOSURE [5], a system that leverages large-scale passive DNS analysis techniques to detect malicious domains. DISCLOSURE [6] is a system that aims at finding C&C servers IP addresses by analyzing NetFlow data (see Section 3.2.3). In sections 3.2.5 and 3.2.4 we present two works, [19] [3], that focus on the infected machines rather than on the C&C servers, an approach that leads to privacy and deployment difficulties. Sharifnya and Abadi [24] propose the interesting approach of grouping together suspicious activities and then look at their history to decide whether they are actually malicious or not (see Section 3.2.6). Haddadi, Kayacik, Zincir-Heywood, and Heywood [12] focus on the detection of automatically generated domains employed by DGA-based botnets, comparing different techniques that do not use *ad hoc* features, but leverage the domain name itself (see Section 3.2.7). Finally in Section 3.2.8 we present PHOENIX [22], a system able to extract clusters of domains related to DGA-based malicious activities from blacklists and a module of CERBERUS.

3.2.1 Detecting Malware Domains at the Upper DNS Hierarchy

Antonakakis et al. [2] with KOPIS were the first to monitor DNS activity at the higher hierarchy level to detect domains related to malicious activities. It leverages this uncharted point of view to explore new features later to be used to train a supervised classifier.

Requester Diversity This group of features capture the geographical diversity of the hosts that query ad domain *d*. Malicious domains are usually queried by machines distributed differently from those querying legitimate domains [22].

Requester Profile This group of features aim at characterizing the different profiles of users that query DNS servers. Especially, it divides them into two broad categories: Those who reside in small networks, for instance a corporate network, and those who reside in large-scale networks. The insight is that while the latter should be more protected, as activity is usually better monitored in such infrastructures, the former are more likely to be exposed to and infected by malware.

Resolved-IPs Reputation This group of features aims to describe whether, and to what extent, the IP address space pointed to by a given domain has been historically linked with known malicious services [2].

Kopis lifetime is divided into *training* and *operation* mode. During the first phase the system is fed with a set of known legitimate and a set of known malware-related domains. For each domain the system computes a feature vector that summarizes the domain behaviour in a time window of m days. Then, during *operation* mode, Kopis builds a feature vector for each unseen domain, capturing its behavior during a given epoch E_j. After the vector is built it assigns a label (*benign* or *malicious*) to the unseen domain and a confidence score.

Limitations

The main limitation of this work is the inability to track down DGA-based botnets, as admitted by the authors themselves. This is caused by the short life span of the AGDs, which makes them untraceable by the detection process implemented by Antonakakis et al. [2]. Moreover, the system is *supervised*, as it requires initial *base knowledge* to be trained with.

3.2.2 Detecting Malicious Activities by Passive DNS Analysis

Bilge et al. [5] in their work propose Exposure, a system that classifies domain names as malicious or benign leveraging large-scale, passive DNS analysis techniques. To achieve their goal, the authors first select 15 features to discriminate between legitimate and malevolent traffic by feeding a J48 decision tree with pre-labeled benign and malicious DNS traffic (i.e., DNS requests and responses). The 15 features are grouped into four logical sets, reported here below.

Time-Based Features capture the peculiar time-related behaviors of malicious domains, for instance querying patterns, as high volume requests followed by a sudden decrease, typical of AGDs.

DNS Answer-Based Features are four features related to the informations that can be retrieved querying a DNS server, as the number of distinct IP addresses resolved by a particular domain d.

TTL Value-Based Features the *Time To Live* expresses how much time, usually in seconds, a cached domain to IP mapping should be considered valid. In their research Bilge et al. [5] found that usually malicious domains feature a low (less than 100s) TTL.

Domain Name-Based Features are two features that aim at capturing the randomness of a domain name.

Then Bilge et al. [5] train a supervised classifier to label domains as malicious or benign, producing a blacklist of domain names–IP addresses, available on their website[1].

[1]`http://exposure.iseclab.org/`

Limitations

Even though able to produce a confirmed blacklist of malicious domains and IP addresses, Exposure is a *supervised* classification system which requires labeled data to be trained.

3.2.3 Detecting C&C Servers by NetFlow data Analysis

Bilge et al. [6] propose Disclosure, a system that aims at finding C&C servers IP addresses by analyzing NetFlow data. The rationale behind this choice resides in the lack of raw network data sources, motivated by administrative and technical issues. NetFlow is a network protocol by Cisco Systems for summarizing network traffic as a collection of network flows [6], where a network flow is a unidirectional sequence of packets that share specific network properties.

Bilge et al. [6] individuate three classes of features that separate benign from malicious network flows. The first one relates to the *size* of the flow. As miscreants' C&C channels have been crafted with the goal of being resilient and stealth, packets belonging to this flows tend to feature a small and constant size. The second concerns the client access patterns. Infected machines will try to contact the botmaster at fixed and regular intervals during the day, while benign traffic exhibits a more "random" behavior. The last class of features captures the *temporal* patterns of access. Legitimate traffic tends to happen during daylight, whilst malicious traffic does not feature this discrimination.

A *Random Forest* classifier is fed with labeled data during the training phase. Then it is tested against data from a university network and from a Tier 1 ISP.

Limitations

Even thought the use of NetFlow data is an interesting and uncharted approach towards the unveil of C&C servers, this approach shows some major shortcomings. Fire [27], Exposure [5] and Google Safe Browsing[1] are reputations systems leveraged by the system to reduce the false positive rate, otherwise unacceptable in volume of points, though low in percentage. Moreover the selected features could be easily circumventable by an attacker. For instance he could tell the bots to communicate during daylight. Or he could instruct them to have a more “random” access behavior.

3.2.4 Detecting FFSN by passive DNS data analysis

FluxBuster was introduced by Perdisci et al. [19] as a system able to detect domains and IP addresses involved in the activity of FFSN, by analyzing DNS data obtained by passively monitoring DSN traffic collected from “above” local RDNSs servers [19]. The authors found four features that characterize DNS traffic belonging to this threat: *i)* a short TTL, *ii)* high frequency in changing the resolving IP addresses, *iii)* high cardinality of the set of resolving IPs [22] and *iv)* the number of networks the IPs reside in. Perdisci et al. [19] use these features to label new data as FFSN and non-FFSN domains using a supervised classifier trained with labeled data. The authors are able to distinguish domains belonging to malicious FFSN from the benign ones, even if such a technique is used also for legitimate purposes (CDN Networks), with a low false positive rate.

[1]`https://developers.google.com/safe-browsing/`

Limitations

Perdisci et al. [19] focus on the *clients*, the infected bots, rather than on the C&C servers of the activities (spam, phishing, etc.). Moreover their approach is *supervised* and requires previous knowledge.

3.2.5 Leveraging NXDOMAIN and clients' IP addresses

Antonakakis et al. [3] propose PLEIADES, a system to *i)* discover and cluster together AGDs that belong to the same botnet, *ii)* build *models* of such clusters and use this knowledge to *iii)* classify unseen domains.

During the first step, *DGA Discovery*, Antonakakis et al. [3] analyze streams of unsuccessful DNS resolutions, as seen from "below" a local DNS server [3]. The streams are collected during a given period of time, and then clustered. The clustering is performed according to two criteria:

- the statistical *similarity* shared by domain name strings;
- the domains have been queries by overlapping sets of hosts [3].

The final output of this module is a set of `NXDOMAIN` clusters: Each cluster is likely to represent a DGA previously unknown or not yet modeled [3]. After this stage is completed the system moves to the *DGA Modeling* phase: This module receives a labeled dataset of malicious and legitimate domains, and leverages this knowledge to train the multi-class *DGA Classifier*.

Then the *DGA Classification* module aims at achieving two tasks. The first one is to label a subset of a cluster of `NXDOMAIN`s with one of the labels seen in the *DGA Modeling* stage. The second is to trace back those domains that belong to the clusters of `NXDOMAIN` but *resolve* to an actual IP address, in order to locate the C&C Server.

Limitations

One limitation reside in the use of a HMM-based detector, unable to detect certain types of DGAs, such as `Boonana`, as indicated by the authors. Another shortcoming, also indicated by Antonakakis et al. [3], is the possible scenario where the attackers produce on purpose random streams of `NXDOMAIN` to sidetrack the detection system. Moreover, once again, PLEIADES requires *i)* the clients' IP addresses and subsequently *ii)* DNS monitoring at the lower level, bringing in all the privacy and deployment-related issues discussed above.

3.2.6 Leveraging activity history to detect botnets

Sharifnya and Abadi [24] developed a system to detect DGA-based botnet based on features that include the history of their activity. They aim at tracking down hosts infected by DGA-based botnets by leveraging DNS queries and trying to group together hosts that exhibit similar malicious behaviors.

The first step is to *whitelist* the DNS queries, filtering out those domains that appear in the Alexa TOP 100 list (list of the most 100 popular domains on the Internet by volume of queries).

Then Sharifnya and Abadi [24] group together those domains that *i)* resolve to the same IP address or *ii)* have the same Second Level Domain (SLD) and TLD. After a time window they label as suspicious those groups where domains are automatically generated. To establish whether a domain is automatically generated, they first compute the distributions of 1-gram and

2-gram for Alexa TOP 1,000,000 sites and the malicious domain names from Murofet [24]. Then they leverage the *Kullback-Leibler divergence* and the *Spearman's rank correlation coefficient* to tell to which category a domain name belongs to.

Simultaneously, another building block of their system, called *Suspicious Failure Detector*, triggers and flags the host as suspicious when a high volume NXDOMAINS DNS queries is originated. Finally, the results from the previous blocks are combined by the *Negative Reputation Calculator*, responsible of the final verdict, i.e., tell whether a host shall be considered bot-infected or not.

Limitations

The idea of grouping together suspicious activities and leveraging their history will be borrowed and, in our opinion, enhanced by CERBERUS. Even in light of the positive results shown by the authors, this work suffers from two major shortcomings. First the system needs a feed of malicious domains automatically generated to compute the malicious distributions of *n*-grams, resulting vulnerable to DGAs that exhibit a new and different distribution of *n*-grams. Second this is an approach *host-based*, which requires the DNS query with the client IP address. This choice involves all the difficulties related to privacy issues and, consequently, leads to non-repeatable experiments [20] and deployment difficulties already discussed in [22].

3.2.7 Using SVM and SSK to classify AGDs

Haddadi and Zincir-Heywood [11] focus on detection of automatically generated domains employed by DGA-based botnets. In their work they compare with other techniques a genetic programming approach, result of their previous work [12], to detect malicious domain names, based only on the string format, i.e., the raw domain name string. This is quite important as to the best of our knowledge, all detection systems in this field analyze DNS network traffic behaviour via classifiers with pre-defined feature sets [11]. We briefly compare the techniques and motivate why we chose the Support Vector Machine approach in CERBERUS.

The first technique to be introduced relies upon Support Vector Machines, state of the art classifiers in many supervised learning tasks. A Support Vector Machine finds a hyperplane that optimally separates the points of the dataset in two classes. This technique can be employed to perform k-classes classification by building k binary classifiers. The machine needs a Kernel Function to be able to separate data which is not-linearly separable. Lodhi et al. [16] proposed the Subsequence String Kernel, a kernel based on common substrings to determine strings' similarity. Once the SVM is equipped with the kernel it must be trained with an initial dataset and then it is ready for classification.

In the authors' Stateful-SBB approach there are three populations that coevolve: A point population, a team population and a learner population. The point population consists of a subset from the training data samples. The learner population represents a set of symbionts, which relate a GP-bidding behaviour with an action [11]. Finally, in the team population we find a set of learners. The evolution follows a Pareto competitive coevolution.

The authors trained the classifier to distinguish between three classes of domains, two malicious and one benign. Conficker and Kraken were chosen as representative of the domains devoted to botnets' C&C communication, while 500 benign domains were manually extracted from the Alexa list. The results favor the SVM approach, which features the highest score (0.996) and a training time (431.53) one order of magnitude less than the SBB approach (2227.64).

Limitations

Haddadi and Zincir-Heywood [11] strongly improve previous features based classifiers, leveraging only the *raw* string of the domain name. Nevertheless their work suffers from being a *supervised* approach. In fact the system must be trained using AGDs blacklist, and it is not capable of detecting new threats. Nevertheless this work lead us to consider the SVM approach when we had to design and implement our classifier, for two main reasons: *i)* in [11] it is the best tool to perform such a task, with performances close to perfection, *ii)* employing the String Subsequence Kernel as system-wide metric to be used by the DBSCAN clustering routine (see Par. DBSCAN Clustering in Section 4.2.3) to perform similarity calculations between domains and cluster of domains. Such matters shall be more deeply discussed in Chapter 4.

3.2.8 Phoenix, Detecting DGA-based botnets

Schiavoni et al. [22] proposed PHOENIX, a system able to extract clusters of domains related to DGA-based malicious activities from blacklists. To this end, PHOENIX first separates domains automatically generated from those created by humans, by leveraging a vector of linguistic features. These linguistic features are *i)* the *meaningful word ratio*, i.e., the ratio of domain's characters composing meaningful words to the cardinality of the domain itself and *ii–iv)* the *popularity score* of the n-grams, with n ranging from one to three. We shall better explain these concepts via an example. Consider the domain names `facebook.com` and `pub03str.info`. Let us compute the *meaningful word ratio* for both of them (see Figure 11).

$d = \texttt{facebook.com}$

$$R(d) = \frac{|\texttt{face}| + |\texttt{book}|}{|\texttt{facebook}|} = 1$$

likely **HGD**

$d = \texttt{pub03str.info}$

$$R(d) = \frac{|\texttt{pub}|}{|\texttt{pub03str}|} = 0.375.$$

likely **AGD**

Figure 11: Meaningful Word Ratio example, Schiavoni et al. [22].

The domain `facebook.com` scores 1, as all of the characters in the domain name contribute to form the words `face` and `book`, which can be found in the English language dictionary. The high score indicates that we have a domain which is likely a Human Generated Domain (HGD). On the other hand, in the domain name `pub03str.info` the only meaningful substring is `pub` and the *meaningful word ratio* is equal to 0.375, likely an AGD.

Consider now the domain names `facebook.com` and `aawrqv.com` and let us compute the popularity score of the 2-gram (see Figure 12).

$d = \texttt{facebook.com}$

fa	ac	ce	eb	bo	oo	ok
109	343	438	29	118	114	45

mean: $S_2 = 170.8$

likely **HGD**

$d = \texttt{aawrqv.com}$

aa	aw	wr	rq	qv
4	45	17	0	0

mean: $S_2 = 13.2$

likely **AGD**

Figure 12: 2-gram score example, Schiavoni et al. [22].

The popularity is computed by counting the number of occurrences of the domain's substrings in the English language dictionary. This score captures the *pronounceability* of a domain name: The more the times a n-gram is found in the dictionary, the higher the score, the easier should be to pronounce the domain name. For instance, the `oo` 2-gram is very common in the English dictionary as it is a common sound in the English language. In our example `facebook.com` features a high score and it is therefore likely to be a HGD, while `aawrqv.com` features a low score and it is therefore likely to be an AGD.

All of the four aforementioned features are combined into a feature vector $\mathbf{f}$, which is computed for every domain belonging to the Alexa Top 100,000 domains list, which lists the 100,000 most popular domains on the web, all very likely to be domain names generated by humans. Then Schiavoni et al. [22] computed the centroid over this group of domains. The idea is that if a domain is farther than a certain threshold λ, then it is likely to be automatically generated.

The algorithm described above is used to separate the HGDs from the AGDs in a blacklist of malicious domains. Then the subset of AGDs is used to generate clusters of domains, depending on the IP they resolve to. Hopefully such IP addresses belong to the C&C servers responsible of controlling the malicious activity the domains refer to. Once the clusters have been generated, PHOENIX uses a list of features to compute clusters' models to be used for classification of unseen domains. The goal is to label unseen domains with one of the threats identified in the clustering phase.

Limitations

There are two major limitations in PHOENIX, the first *conceptual* while the second concerns the system's validation. The *conceptual* limitation resides in the features used to classify unseen domains. One of these features is the IP address of the C&C servers, which means that if an unseen domain does not share the IP address with one of the clusters it is not considered malicious. Obviously this is not true, as actually attackers do change the location of the C&C server once they are identified. Moreover, other two features used by Schiavoni et al. [22] are the length of the domain and the TLD. This means that if an attacker decides to use a DGA that makes domains longer or shorter than usual, or domains that exhibit a different TLD, PHOENIX does not label them as belonging to the threat they actually belong to. The other limitation has to do with how the system was validated. Schiavoni et al. [22] did not test PHOENIX in the wild, whereas the purpose of a detection system is to detect threats analyzing real world data.

3.3 Goals and Challenges

Our primary goal is to isolate in an automatic and unsupervised fashion DGA-based botnets, clustering malicious activities that use the same DGA, by analyzing DNS passive data, in order to unveil botnets' C&C servers IP addresses, as to make it possible to apply the required countermeasures.

Most systems described above, [11] [24] [2] [3] [19] [6], suffer from the *supervised* approach. We think that in this particular scenario this is a major issue: We would like our system to be able to detect *new* threats with little or no previous knowledge. Other systems, [3] [24], leverage the *clients'* (i.e., the *bots'*) IP addresses to cluster together DNS traffic that relates to the same malicious activities. This approach falls into the privacy related issues that arise when dealing with this kind of data. Moreover the monitors to get these samples must be located at the lower levels of the DNS hierarchy, which causes difficulties in the deployment of the monitors themselves. One system, Pleiades [3] does not perform well with some types of DGAs (e.g., `Boonana`), whereas we want to build a detection software that does not leverage *some* features of peculiar DGAs.

Phoenix [22] is the system that meets most of our criteria, as it uses an *unsupervised* approach and analyzes passive DNS data, free of any privacy issues. Still, it shows some major shortcomings, as the impossibility of detecting unknown botnets, the fact that the classifier is not resilient to small modifications in the DGAs and the lack of a validation in the wild, which makes us agnostic with respect to Phoenix's effectiveness once deployed in the real world.

Building a system that evolves in an automatic and unsupervised fashion constitutes a hard challenge to face. We have to find a way to filter out benign domains while keeping the malicious domains that does not require knowledge to be fed, i.e., we have to think about general features that characterize malicious AGDs. Moreover this filtering process must be fast enough to allow an online detection process while dealing with high volumes of data. We also need to understand how to cluster together domains that look "similar", therefore we have to develop a concept of similarity between domain names (i.e., strings) that is able to capture the patterns shared by AGDs produced by the same DGA. Once the malicious activities are clustered, we need to understand how we can use this knowledge to build a classifier that is resilient to small changes in the DGA and does not use *ad hoc* features. The previous challenges brought to the birth of CERBERUS, which shall be presented in Chapter 4.

CHAPTER 4

INTRODUCING CERBERUS

Cerberus or Kerberos () is a three-headed guardian dog from the Greek mythology, in charge of patrolling the gates of Hades, land of the dead. Preventing dead souls to evade the underworld was his duty. In the same fashion we wanted to build a tool to *detect* malicious domains employed by botnets and track down the IP addresses these domains resolve to, as to guarantee that they shall not tarnish the namespace ever again.

As the hellhound featured three heads, our CERBERUS operates in three phases: The **Bootstrap Phase**, where the initial optional ground truth is generated, the **Filtering Phase**, where the DNS data is filtered, and the **Detection Phase**, where domains are classified and new threats are discovered.

Chapter Organization

The remainder of this chapter is organized in the following fashion:

- in Section 4.1 we will describe CERBERUS at a high level of abstraction, providing an overview of the overall process;
- in Section 4.2 we will tackle all the elements that form CERBERUS.

4.1 Cerberus Overview

Cerberus is composed of three main modules or phases: **Bootstrap**, **Filtering** and **Detection** (see Figure 13). The **Bootstrap Phase** is fed with a blacklist of malicious domains and provides the initial ground truth to the system in an automatic and unsupervised fashion. We use the term *ground truth* to refer to the knowledge that the system trusts even though its correctness was not validated by any external authority. The ground truth consists of a set of clusters of domain names and the IP addresses they resolve to, clusters that represent DGA-based malicious activities employing the same DGA. This knowledge is used by the **Detection Phase**.

The **Detection Phase** consumes the output of the **Filtering Phase**, where a stream of DNS replies is filtered to obtain a list of likely malicious domains. To this end we employ a list of filters mostly based on heuristics and whitelists to reduce as much as possible the number of legitimate domains, while keeping the suspicious ones. After the **Filtering Phase** is completed, the Classifier uses the ground truth previously generated to *detect* known threats, i.e., the Classifier returns a list of malicious domains labeled with one of the clusters from the ground truth. To this end, for each unseen domain it selects those clusters that share the domain IP address and then decides to which one it should be assigned using a Support Vector Machine that leverages the Subsequence String Kernel to compute similarity between strings.

Those domains that do not share their IP address with the ground truth are fed to the **Time Detective**: The rationale is that at this very moment we cannot establish whether these "suspicious" domains are actually malicious, but by analyzing the activity related to their IP

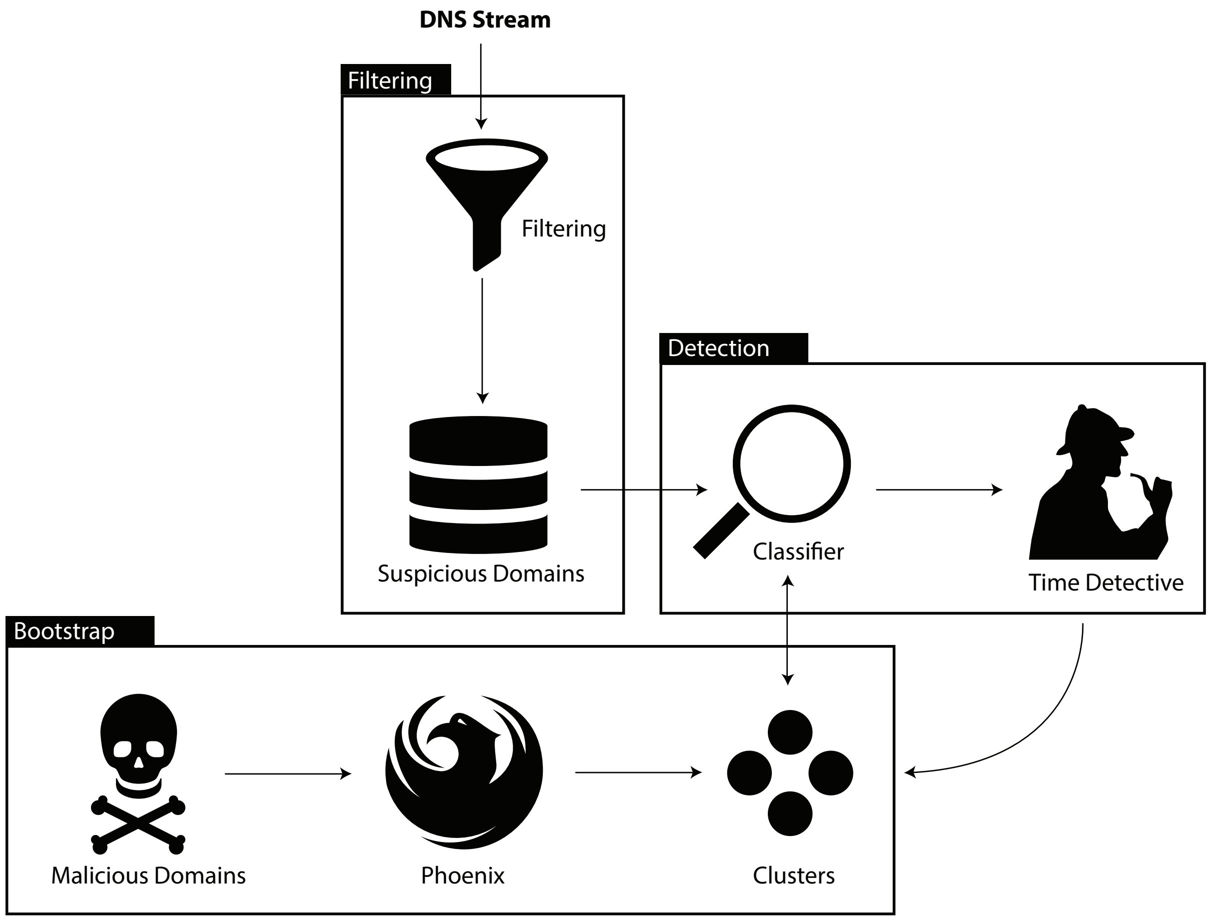


Figure 13: Cerberus overview.

addresses we may be able to correctly identify the threats. The **Time Detective** stores the "suspicious" IP addresses and keeps track of the domains that resolve to them throughout time. After Δ time has passed it groups them by their Autonomous System, and clusters together the domains that are deemed similar (see Section 4.2.3). After the new clusters are generated they are added to the ground truth and this further knowledge is then used to increase the range of detected threats.

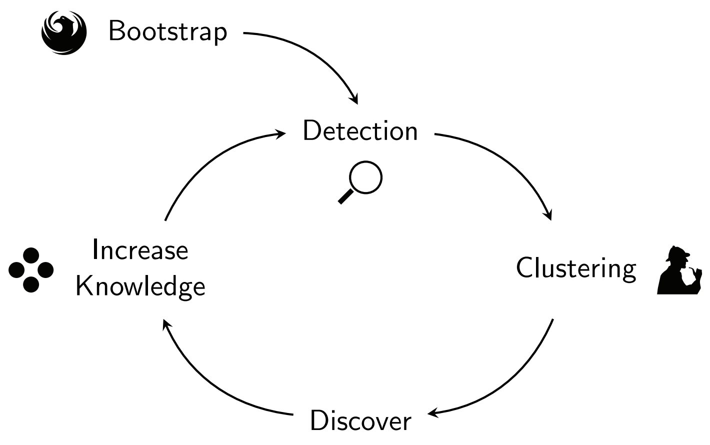


Figure 14: Cerberus lifecycle.

This process is summarized in Figure 14: Cerberus can automatically evolve by increasing its knowledge in an unsupervised fashion: The Clustering routine allows to discover new threats that are added to the ground truth. Then the system can return in the **Detection Phase**, leveraging the new knowledge. In the next section we discuss in further detail what briefly described above.

4.2 Cerberus Details

In the previous section we have given a brief overview of how CERBERUS works. In the following of this section, we shall go in deeper detail, throughly explaining every aspect of the **Bootstrap**, **Filtering** and **Detection** phases.

4.2.1 The Bootstrap Phase

The **Bootstrap Phase** aims at providing CERBERUS with the initial ground truth. This knowledge is not mandatory to have, as the system needs no *a priori* knowledge to start detecting new threats. Nevertheless it is useful to have the possibility to plug it into CERBERUS, which can use it to label the unseen domains during the **Detection Phase**. To this end we employ PHOENIX. PHOENIX takes as initial input a list of malicious domains. It then applies a DGA filter (the same used in Paragraph 4.2.2) to filter out those domains that appear likely benign, under the assumption that domains generated manually, by a human, are benign, in contrast to AGDs, which are generated automatically by a DGA. Therefore the remaining domains are both *non-benign* (i.e., suspicious) and *automatically generated*.

CERBERUS clusters these domains depending on the IP address(es) they resolve to. Hence at the end of the clustering process, CERBERUS has a list of clusters of domains and the IP addresses they resolve to, domains produced by the same DGA and IP addresses that should belong to the C&C servers, and it will later leverage this list to label unseen domains in the **Detection Phase**.

The following are two of the eleven clusters generated by the **Bootstrap Phase**:

Cluster f105c	
Threat:	Palevo
IPs:	`176.74.176.175`
	`208.87.35.107`
Domains:	`cvq.com`
	`epu.org`
	`bwn.org`

Cluster 0f468	
Threat:	Sality
IPs:	`217.119.57.22`
	`91.215.158.57`
	`178.162.164.24`
	`94.103.151.195`
Domains:	`jhhfghf7.tk`
	`faukiijjj25.tk`
	`pvgvy.tk`

4.2.2 The Filtering Phase

After the **Bootstrap Phase**, Cerberus aims at classifying unseen domains from a stream of DNS passive data. The DNS stream we receive comes from the real world, hence it features both benign and malicious domains. Therefore we want to reduce as much as we can the amount of legitimate domains while keeping as many malicious domains as possible. To achieve this goal the DNS feed goes through a list of filters listed here below.

Alexa Whitelist

This filter removes all the domains that rank amongst the first 1,000,000 in the Alexa Top Domains list. The rationale is that the most popular domains in the Web are less likely to be malicious. Even though this is not completely true (as of 2012, BarracudaLabs found 58 drive-by-download malicious websites in the first 25,000 Alexa domains[1]), there are no known cases of malicious AGDs found among the Alexa Top 1M. Moreover we are looking for domains used to set up the C&C channel: They are usually valid for one day and then thrown away and it is very unlikely that they make it to the list.

Content Distribution Networks

DGAs are used to legitimate ends by Content Delivery Networks (CDNs). These infrastructures consist of a large distributed system of servers, deployed with the intention of providing users with content at high availability and high performance. YouTube and Amazon CloudFront are examples of this type of networks. In our system we filled in a list of the most popular CDNs: If a domain belongs to one of them it is filtered out. For instance, the domain `r4---sn-a5m7lnes.c.youtube.com` belongs to the YouTube CDN: It was clearly generated by some DGA, but it is not malicious and therefore it must be filtered out.

Top Level Domain Whitelist

Some TLDs require clearance before the domain can be registered (see Table III). When the attackers register their domains they do not want to go through authorizations by third party authorities, for two obvious reasons: *i)* they do not want anyone to investigate the domain they are going to register and *ii)* they cannot wait for the time required by clearance process. Therefore all of those domains that feature a TLD listed in Table III are filtered out.

[1]`https://media.blackhat.com/ad-12/Royal/bh-ad-12-quanitfying-royal-slide.pdf`

TABLE III: TLDS REQUIRING CLEARANCE BEFORE REGISTRATION.

TLD	Entity	TLD	Entity
`.ac.uk`	British academic domains	`.edu`	educational
`.aero`	air-transport industry	`.gov`	governmental
`.arpa`	Address and Routing Parameter Area	`.int`	international organizations
`.coop`	cooperatives	`.mil`	US Military
`.museum`	museums	`.pro`	professions
`.post`	postal services	`.travel`	travel and tourism industry

Time To Live

The TTL parameter sets for how long a cached DNS record is to be considered valid in a local DNS Server. Previous works in literature ([5], [13]) stated that this value was set to very low values (100s) in the case of malicious domains. The rationale, from the botnet operators, is that these infrastructures need to update fast their C&C locations and therefore need the DNS server to update the records at the same pace. However, more recent investigations seem to state the opposite. Xu and Wang [29] from Palo Alto Networks reported that approximately 80% of fast-flux domains exhibit a change rate greater than 30 minutes/IP, while in 2008 Holz et al. [13] reported that fast-flux domains exhibited a changing rate less than 10 minutes/IP. This change, and the subsequent change in TTL values, is due economic reasons and to avoid detection based on changing rate. Therefore we decided to filter out all the domains featuring a low TTL value, less than 300 seconds. We chose this value after analyzing two weeks of

DNS data from the ISC/SIE dataset in our possession: We manually searched for the domains classified as malicious by an early version of Cerberus, and noticed that all the benign domains featured a TTL equal or lower than 300 seconds.

Phoenix DGA Filter

The domains employed in DGA-based malicious activities feature that randomness captured by Schiavoni et al. [22] in Phoenix. Therefore we filter out those domains that do not exhibit the required level of randomness to be labeled as AGDs.

Whois Queries

Finally we look at the registration date. We leverage the insight that attackers do not (and sometimes *cannot*, as when using unpredictable seed in the DGA, e.g., Twitter trend topic) register their domains much time in advance, as otherwise they would not be able to leverage the resilient migration strategy allowed by the use of DGAs: If the C&C is unveiled they should update the DNS records for all the pre-registered domains.

Therefore we query a `Whois` server to ask for the registration date of every domain: If the domain is “old” it is discarded, whereas if either it was recently registered or the `Whois` server replied with an error code (e.g., no information is available for that domain), we conservatively keep the domain. For all the retained domains we store the registration date for later use in the detection phase.

Summary

We want to recap here the features of the domains that remain at the end of the filtering process and that constitute the input for the **Detection Module**. For each step we report also the number of domains that persist after the filtering process in a pilot experiment conducted on 30' of sampled traffic (see Section 6.2).

20,000 domains remain with a TTL greater than 300 seconds;

19,000 domains not in the Alexa Top 1M whitelist;

15,000 domains not in the most popular CDNs;

800 domains likely to be AGDs;

700 domains featuring a TLD that does not require previous authorization;

300 domains that have been recently registered.

4.2.3 The Detection Phase

In the **Detection Phase**, we aim at recognizing domains belonging to known botnets and to discover new ones. These two tasks are achieved by two different modules: The **SSK Classifier** and the **Time Detective** respectively.

The Subsequence String Kernel Classifier

This component is fed with the data coming from the **Filtering Phase**. It classifies each unseen domain domain d using the knowledge provided by PHOENIX, i.e., the clusters of DGAs, eventually including those generated by CERBERUS. The classifying process consists of assigning d to one of those clusters. Schiavoni et al. [22] had developed their own classifier, but it suffered from the major shortcoming of using *ad hoc* parameters, such as the domain length and the TLD. This means that if d features a TLD other than those in the clusters, or if it features a length that does not reside in the previously seen boundaries, it is discarded.

We wanted to have a classifier resilient to such changes, one that would label d leveraging only the *string* itself as a feature to compute similarity amongst d and the clusters. Haddadi et al. [12] compared different approaches in classifying AGDs: As the Support Vector Machine technique outplayed all the others with respect to the F-Score, we decided to use this approach.

Support Vector Machines

The Support Vector Machine is a binary learning machine that can be used for classification and rule regression [1]. The main idea of this classification algorithm is to build a hyperplane that optimally separates the samples of data into two categories with maximal margin [12]. It is possible to build a k-classes classifier, with k greater than two, by building k binary classifiers.

Support Vector Machines must undergo a *training phase*, where they are fed with points in the form of $\{\mathbf{x_i}, y_i\}$, where $\mathbf{x_i}$ is a feature vector and y_i is the binary class label (e.g., 1 or -1). Note that in our context, the labels need not to be provided manually because we use clusters as labels, which are generated automatically. There is a parameter to be set during the classification process: C, which controls the punishment function for misclassified points. In our implementation we left this parameter to 1, its default value.

When dealing with linear data we have to find two hyperplanes:

$$\mathbf{w} \cdot \mathbf{x} - b = 1$$

and

$$\mathbf{w} \cdot \mathbf{x} - b = -1$$

that bound a region, called *the margin*, where no data point is present. Training the SVM in this case means to maximize the distance between the hyperplanes. SVMs can be also employed to classify data that are non-linearly separable, but they need to be equipped with a *kernel function*, a non-linear mapping of an input data into a high dimensional feature space [12]. When the mapping is completed, the hyperplane that maximizes the separation margin can be constructed. Then, as a final step, a linear mapping from the feature space to the output space is required [12]. In the next paragraph we shall better explain what a kernel function is and introduce the Subsequence String Kernel.

Subsequence String Kernel

A function that calculates the inner product between mapped examples in a feature space is a kernel function [16]. Therefore for any mapping:

$$\phi \,:\, D \rightarrow F$$

we define a kernel function as

$$K(d_i, d_j) = \langle \phi(d_i), \phi(d_j) \rangle$$

where $\langle \cdot, \cdot \rangle$ is the dot product and d_i, d_j are feature vectors. In 2002 Lodhi et al. [16] proposed a kernel function to measure the similarity of strings. The idea is to compare them by means of the substrings they contain: The more substrings in common, the more similar they are [16]. We can better explain the process by looking at the example in Table IV.

TABLE IV: SSK EXAMPLE FROM LODHI 2002.

	c-a	c-t	a-t	c-r	a-r
ϕ(cat)	λ^2	λ^3	λ^2	0	0
ϕ(car)	λ^2	0	0	λ^3	λ^2

$$\ker(car, cat) = \lambda^4$$
$$\ker(car, car) = \ker(cat, cat) = 2\lambda^4 + \lambda^6$$
$$\ker(car, cat) = \frac{\lambda^4}{(2\lambda^4 + \lambda^6)} = \frac{1}{(2 + \lambda^2)}$$

We consider the two strings `car` and `cat`. The first step is to compute the feature space of the strings to be compared. In this case it is a 5-dimensions space composed by the features `c-a`, `c-t`, `a-t`, `c-r` and `a-r`, which are all the possible non-contiguous two characters long substrings of `car` and `cat`.

We then set a *decay factor* λ that measures a decay in the quality of the similarity between strings, i.e., if we have λ^n, it means that we need n characters of the string to match the substring. In our example we have for instance `c-t`, which is present in `cat`, but it requires three characters to be matched: `c`, a and `t`. That is why we then find λ^3 in the cell (ϕ(cat), `c-t`).

We then define the inner product between two strings s and t as the sum over all common subsequences weighted according to their frequency of occurrence and lengths [16]. In formulas:

$$K_n(s,t) = \sum_{u \in \sum^n} \langle \phi_u(s) \cdot \phi_u(t) \rangle = \sum_{u \in \sum^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}$$

In our example `car` and `cat` share only the substring `c-a` (we have highlighted the column). Therefore we have:

$$\mathrm{ker}(car, cat) = \lambda^2 + \lambda^2 = \lambda^4$$

which is the unnormalised kernel. To get the normalised kernel we divide by $\mathrm{ker}(car, car) = \mathrm{ker}(cat, cat) = 2\lambda^4 + \lambda^6$, which yields $\frac{1}{(2+\lambda^2)}$.

The Classification Process

When an unseen domain d arrives, we select the clusters that share the IP address with d. Those clusters are used to train the Support Vector Machine that is then used to classify d. What happens when no cluster shares the IP address with d is explained in the next paragraphs.

The Time Detective

When an unseen domain d does not share the IP address with the clusters of the ground truth, we issue the **Time Detective**: We want to keep track of the activity related to its IP address for Δ time to see if it shows hints of maliciousness.

After Δ time has passed, first we group together domains that resolved to IPs that reside in the same Autonomous Systems, then we try to extract clusters of similar (similarity is computed using the SSK) domains using the DBSCAN clustering algorithm. After the clustering routine is done, we add these new clusters of malicious domains to the ground truth. To see whether two clusters (one from the new ones and the other from the old ones) are similar, we run a similarity test. After this stage is completed, CERBERUS starts classifying new unseen domains, leveraging the increased knowledge. In the next paragraphs we shall tackle every aspect of the aforementioned process.

DBSCAN Clustering

The DBSCAN algorithm relies on the concept of *density reachability.* In Figure 15 B is density-reachable by A, as there is a chain of data points such that the next point is no farther from the previous one than ε. DBSCAN allows several metrics to measure the distance among samples: Euclidean, Manhatthan, Jaccard and Mahalanobis are a few examples. In Cerberus we use a Kernel Distance and we compute the distance matrix using the Subsequence String Kernel. ε is a parameter set *a priori*, usually by looking at the k-distance graph. The other parameter DBSCAN requires to be set is *minPts*, the minimum number of points required to form a cluster. For instance, in our situation, if *minPts* was set to four, all the points from A to B would belong to the same cluster. If it was set to six, all the points from A to B would be considered noise. In our implementation we set the *minPts* parameter to Δ, where Δ measures in days how much time we wait before performing the clustering routine. We chose this number following this rationale: In Δ time a cluster must count at least one domain for each day passed, i.e., the bots must have contacted the C&C Server at least once a day. In the next paragraph we will explain how we set the ε parameter.

ε estimation

To estimate ε we used a heuristic that leverages the *k distance* graph. The *k distance* graph is a graph in which two vertices p and q are connected by an edge, if the distance between p

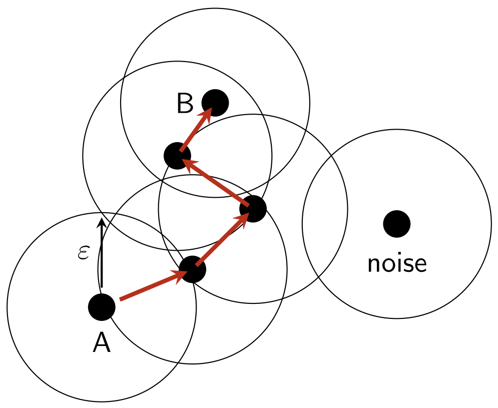


Figure 15: The DBSCAN Algorithm.

and q is among the k-th smallest distances from p to other objects of the population[1]. We set k equal to *minPts*, and we calculate the variance of the distances for the neighborhood of each point. We then consider the point that features the neighborhood with the lowest variance. From that neighborhood we take the highest distance. The rationale is that we want to be sure that the *minPts* least dispersed points are clustered together.

It could happen that this value needs to be slightly adjusted. Therefore we modify ε using a tolerance τ parameter. In formulas:

$$\varepsilon = \varepsilon \times \tau$$

[1] `https://en.wikipedia.org/wiki/Nearest_neighbor_graph`

We try different values of τ that range from -0.5 to 0.5, step 0.1, and we choose the value that yields the best clustering quality. In the next paragraph we explain how we measure the clustering quality.

Measuring Clustering Quality

To measure the clustering quality we use the C-Index proposed by Hubert and Schultz [14]. They define N_W as the sum of the number of pairs in each cluster:

$$N_W = \sum_{k=1}^{K} \frac{n_k(n_k - 1)}{2}$$

where n_k is the cardinality of cluster k. They then define S_W as the sum of the N_W distances between all the pairs of points inside each cluster, $S_{\min}$ as the sum of the N_W smallest distances between all the pairs of points in the entire dataset and $S_{\max}$ as the sum of the N_W largest distances between all the pairs of points in the entire dataset.

Hubert and Schultz [14] define the C-Index as the ratio of the sum of distances infra-cluster to the sum of the distances intra-clusters (normalized). The C-Index ranges from 0 to 1, and lower values yield better clusters. In formulas:

$$C = \frac{S_W - S_{\min}}{S_{\max} - S_{\min}}$$

As previously stated we iterate the clustering process for each list of domains over eleven values of tolerance (from -0.5 to 0.5, step 0.1) and keep the clusters that feature the lowest C-Index. Once the process is completed for all the Autonomous Systems, we have to decide whether the new clusters must be *added* to the ground truth produced by PHOENIX, or if there are couples of clusters that, though not sharing the IP addresses, should be *merged* together as they represent the same family of DGA. In the next paragraph we explain how we measure similarity between clusters.

During the test described in Section 6.4, When applying the clustering routine to AS 22489,

$$\tau = 0.2$$

yielded the lowest value of C-Index = 0.0032.

Measuring Clusters Similarity

CERBERUS is able to tell when two clusters are to be merged together. To this end it employs the Welch's test.

Suppose we want to establish whether cluster A should be merged or not with cluster B. We can represent both of the clusters in matrix form, where we have the elements on rows and columns, and a cell contains the *distance* between the elements, computed using the SSK.

$$A_{m,m} = \begin{array}{c|cccc|} & a_1 & a_2 & \cdots & a_m \\ a_1 & a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_2 & a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_m & a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{array} \qquad B_{n,n} = \begin{array}{c|ccc|} & b_1 & \cdots & b_n \\ b_1 & b_{1,1} & \cdots & b_{1,n} \\ b_2 & b_{2,1} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_n & b_{n,1} & \cdots & b_{n,n} \end{array}$$

We then build the $n \times m$ matrix $A \sim B$, which features the n elements of A as rows and the m elements of B as columns.

$$A \sim B_{m,n} = \begin{array}{c} \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \cdots & b_n \end{array} \\ \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m,1} & d_{m,2} & \cdots & d_{m,n} \end{pmatrix} \end{array}$$

We then run the Welch's test: The Welch's test is used when two samples may show different variances and it is a variation of Student's *t-test*. We leverage the two-tailed test, where the null hypothesis states that the two samples have equal mean. The test will yield a *p-value* that will allow to accept or reject the hypothesis. In CERBERUS we decided to set the *p-value* to 1%: Values greater than that will tell the clustering routine that there is not enough statistical evidence to keep the clusters separated, and CERBERUS will merge the clusters together.

The rationale behind this procedure is that when two clusters actually are the same cluster, if we mix the elements, the original cluster (A in our example) and the "mixed" one ($A \sim B$) should feature the same distribution of the distances.

Cluster a and b Welch's test yielded the values

$$t \simeq 2.2 \quad \text{and} \quad p \simeq 0.03$$

As we set the *p-value* to 1%, we cannot refuse h_0, i.e., the clusters are merged together. Further investigation on `palevotracker.abuse.ch` confirmed that both IP sets identify Palevo C&C (see Section 6.4).

CLUSTER A		CLUSTER B	
IPs:	`176.74.176.175`	IPs:	`82.98.86.171`
	`208.87.35.107`		`82.98.86.176`
			`82.98.86.175`
			`82.98.86.167`
			`82.98.86.168`
			`82.98.86.165`
Domains	`cvq.com`	Domains	`knw.info`
	`epu.org`		`rrg.info`
	`bwn.org`		`nhy.org`

4.3 Summary

In this section we have introduced CERBERUS and explained how it works. CERBERUS lifecycle is composed of three main phases. During the **Bootstrap Phase** it produces the initial ground truth. When we use the term *ground truth*, we do not mean ground truth *per se*, as there is no manual check of the results produced, but we mean the knowledge that the system trusts. This knowledge is later used in the **Detection Phase**, but before the actual detection can start CERBERUS goes through the **Filtering Phase**. During the filtering phase the DNS stream to be analyzed is sifted out from legitimate domains, using a series of heuristics.

The remaining data is fed to the aforementioned **Detection Phase**, during which Cerberus uses the clusters of AGDs from the initial ground truth to label the unseen domains. Those domains that do not share the IP address with the clusters are recorded for Δ time and then a clustering routine is performed. The newly generated knowledge is then added to the system's ground truth. In the next chapter we describe the implementation of Cerberus, focusing on the crucial parts of the system.

CHAPTER 5

SYSTEM IMPLEMENTATION

In this chapter we discuss a few aspects concerning the implementation of Cerberus. Cerberus is implemented in `Python`, and it uses well known and high performance libraries for machine learning and scientific calculus. We discuss the system's architecture, focusing on the process and the actors involved in the unseen domains' classification and new threats' discovery. Moreover, we describe in deeper detail a few core aspects of Cerberus, as the possibility of customizing the system using an external `json` configuration file and how we managed to address the performance challenges of the **Filtering Phase** leveraging the concepts of *map* and *reduce*.

Chapter Organization

The remainder of the chapter is organized in the following fashion:

- in Section 5.1 we discuss the architecture of Cerberus, describing in detail the system's lifecycle and the actors that participate;
- in Section 5.2 we discuss the actual implementation of the core parts of the system.

5.1 System Architecture

In this section we describe Cerberus's architecture in a *top down* fashion, first introducing the *process*, i.e., how the system analyzes the DNS data and detects botnets, and then focusing on the *actors*, the *classes* and *modules* that compose Cerberus. Before that we will provide a few technical details about the technologies employed.

Cerberus is implemented in `Python`, version 2.7.3, and it uses `MongoDB`, version 2.4.9, as database system, as it offers good performances. We used `Shogun`, `SciPy` and `scikit-learn` to implement the SVM with SSK classifier and the DBSCAN with SSK clusterer, while `NumPy` offered the high-performance data structures used in the system.

5.1.1 The Process

Cerberus needs to have two folders and a configuration file to function. The first folder serves as a buffer where an external service (e.g., a passive DNS monitor), sends the DNS data, which must be organized into text files where each line is a DNS reply in `json` format. The configuration file contains the system's settings, later to be thoroughly explained (see Section 5.2.1).

After γ time (e.g., one hour), the input files are moved to a second folder to start the analysis process (❶), while in the first folder, now emptied, the buffering process can start over again. The DNS replies are parsed and a list of `Domain` objects (see Section 5.2.2) is created and stored into the database (❷). This list shall later be referenced to as the `dirty_dns` list.

Once the previous storing procedure is completed, the filtering routine can start. The domains from the `dirty_dns` list are loaded in memory from the database and filtered (❸) applying the list of filters described in Section 4.2.2. The resulting list of domains, to be referred to as `cleaned_dns`, is then stored into the database (❹).

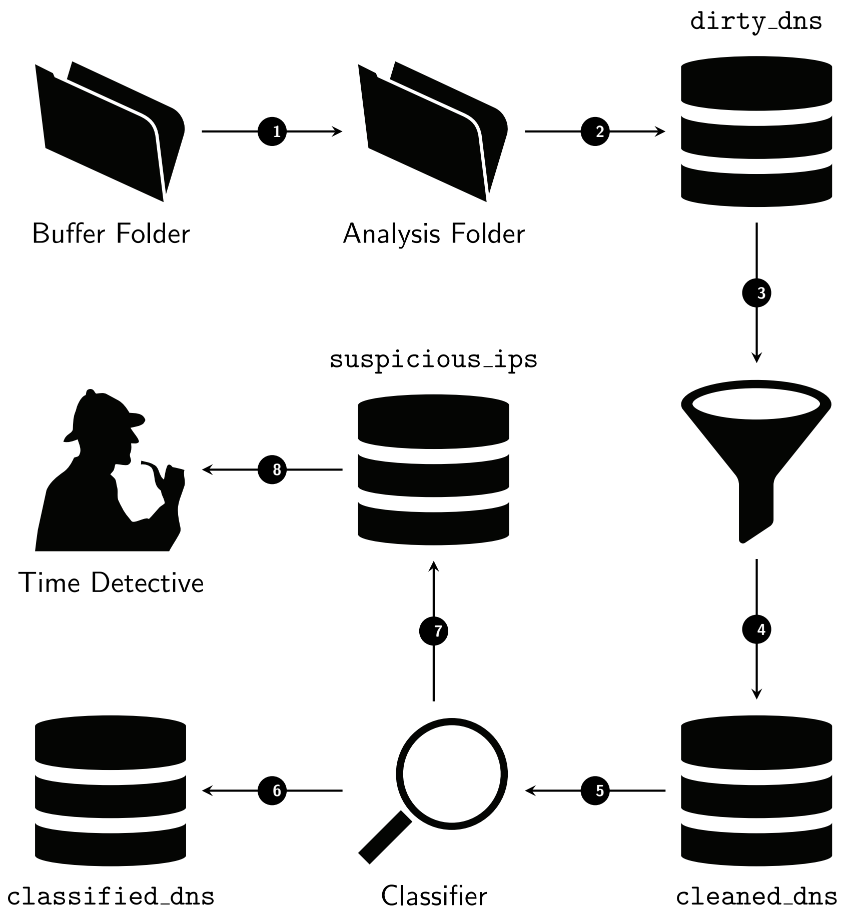


Figure 16: Cerberus classification process.

Now the classification process can start. For each domain from the `cleaned_dns` list, Cerberus first looks at the IP address and sees whether one or more clusters from the ground truth share the same IP address (❺). In this case, a SVM is trained and a label is assigned to the domain: All the domains for which it was possible to assign a label are stored in a database schema called `classified_dns` (❻). The domains that do not share their IP address with the ground truth are not discarded: Their IP address is stored in the `suspicious_ips` database schema (❼). In this collection each document is composed of *i)* the IP address and *ii)* the domains that resolved to that IP address throughout time.

After Δ time, the clustering routine tries to generate new clusters to be added to the ground truth (❽). The IPs collected in the `suspicious_ips` schema are grouped by their respective Autonomous System, and their lists of resolved domains are merged together. Then, for each AS, we run the DBSCAN clustering routine on the list of domains, which produces the aforementioned new clusters of similar domains. For instance, say we have IP Φ that resolves the domains $\mathbb{D}(\Phi) = \{\phi_1, \dots, \phi_n\}$, IP Ω that resolves the domains $\mathbb{D}(\Omega) = \{\omega_1, \dots, \omega_m\}$, and for the both of them the AS is α. Then Cerberus will group together Φ and Ω, and will merge their lists of domains, obtaining

$$\mathbb{D}(\Phi, \Omega) = \{\phi_1, \dots, \phi_n, \omega_1, \dots, \omega_m\}$$

that is then fed to the DBSCAN clustering routine to extract the clusters of similar malicious domains. When this stage is terminated, CERBERUS adds these clusters to its ground truth and restarts analyzing the DNS stream, leveraging the enhanced knowledge.

Before describing the actors involved in the aforementioned process, we would like to justify the heavy use of the database, i.e., why we chose to store the domains at each step. The reason is to have a system *failure compliant*: If CERBERUS crashes, it is possible to retrieve the results of the last completed stage, instead of wasting time in starting the classification from the beginning. For instance, if the system fails right after the filtering process, the list of `cleaned_dns` domains can be retrieved from the database and classified.

5.1.2 The Actors

CERBERUS is organized into six major classes (see Figure 17) of which the homonymous `Cerberus` class is the main one, exposing the high level functionalities of the system. It is initialized with a `config` parameter, a `json` file that contains the variables to customize the system at the user's needs. The `bootstrap()` function initializes the system with the ground truth and it is automatically issued before the system starts to classify the unseen domains.

The `fetch_dns_cleaned_stream()` and `classify_domains()` subroutines are issued by the `execute_daily_routine()` function, which takes care of classifying the unseen domains coming from the DNS data. They are both executed by calls to the `DataManager` class, which offers lower-level functionalities to *i)* parse the input file of DNS data, *ii)* filter the DNS data and *iii)* classify the DNS data. While the parsing is achieved by the class itself, filtering and classification are relayed to other modules. The `SVMSSKClassifier` takes care of selecting the

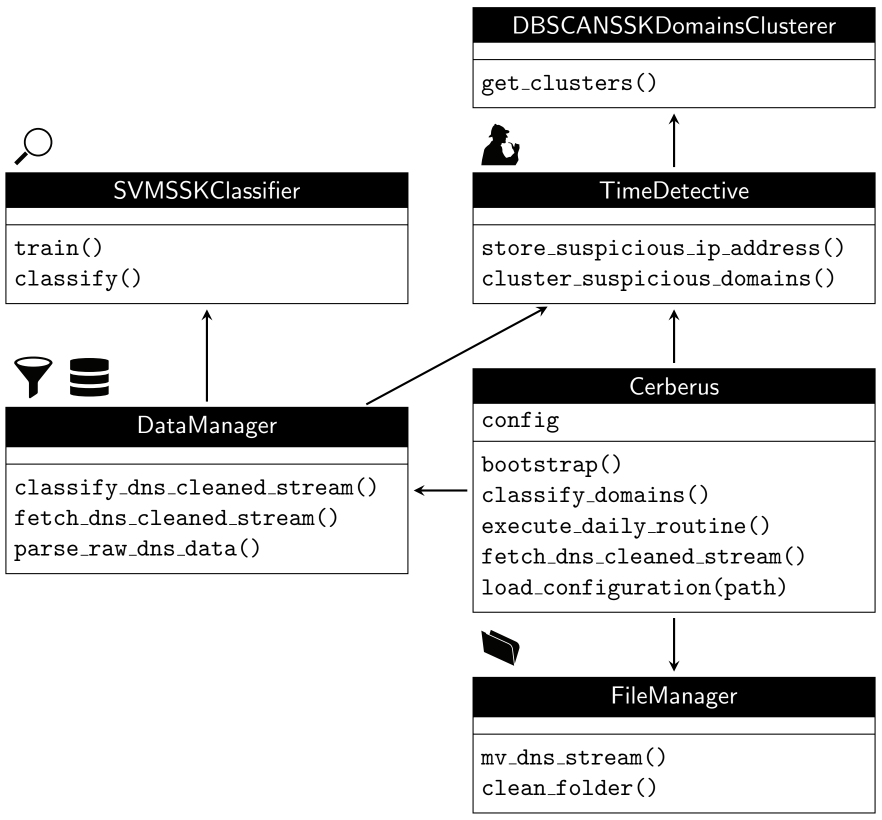


Figure 17: Cerberus' class diagram.

candidate clusters (those that share the IP address(es) with the unseen domain about to be classified), train the SVM and classify the unseen domain. The filtering process is achieved by the `stream_filter.py` module, described in Section 4.2.2. It is not reported in Figure 17 as it is not a class.

The `TimeDetective` class is used by both the `DataManager` and the `Cerberus` class. In the first case it is called the `store_suspicious_ip_address()` method, which takes care of keeping track of the suspicious IPs and storing them in the `MongoDB` database. In the second case, `cluster_suspicious_domains()` is issued, which in turn calls the `get_clusters()` method, which generates the clusters of AGDs and belongs to the `DBSCANSSKDomainsClusterer` class. Finally, the `FileManager` class handles the routines related to filesystem calls.

5.2 System Details

In this section we focus on the core aspects of the implementation of Cerberus, and provide a more detailed description. First we present the *configuration file* used to customize the system to user's needs. Than we briefly see the `Domain` and `DomainName` classes, used by Cerberus to store the data throughout the analysis. Finally we tackle the crucial aspects of the three phases of Cerberus: **Bootstrap**, **Filtering** and **Detection**.

5.2.1 The Configuration File

Cerberus was designed to be flexible and customizable. To this end, we envisaged the possibility of setting a few variables via an external `json` file. The user needs to put this file in the same folder where Cerberus is and name it `config.json`.

`Cerberus`, the main class, is then initialized with it and the values are passed to the various manager classes. An example is reported in Listing 5.1 where we explain the meaning of the different variables.

Listing 5.1: Cerberus' configuration file.

```
{
    "database_type": "mongodb",
    "dns_path": "dns_traffic",
    "filters": [
        "ttl",
        "alexa",
        "cdn",
        "ns",
        "tld",
        "dga",
        "whois"
    ],
    "reports_path": "reports",
    "stream_path": "dns_stream",
    "ttl": 300
}
```

- `database_type` is the database system used: Cerberus features a layer of abstraction to access the data that allows other database systems to be used;

`dns_path` is the path of the folder where the DNS data is moved before being analyzed;

`filters` is the array of filters to be applied in the filtering phase. The user can decide which filters are to be used and the order they appear in the file is the same order in which they shall be applied;

`reports_path` is the folder where `Cerberus` stores the database dumps after a classification routine is completed;

`ttl` is the TTL threshold to be used when filtering out the domains exhibiting a TTL value lower than the set threshold;

`dns_stream` is the path of the folder where the DNS data is collected, i.e., the *buffer* folder;

5.2.2 The `Domain` Data Structure

DNS records are parsed into `Domain` objects, previously implemented by Schiavoni et al. [22] in PHOENIX, and still used in CERBERUS to ensure inter-operability. `Domain` objects have two main attributes the `domain_name`, instance of the `DomainName` class, and the `ip_mappings`, both private and accessible via getters (see Figure 18).

The `DomainName` contains the domain name, parsed in labels. It offers access to the whole original domain name, to the TLD and to the chosen prefix respectively via the `get_original_-domain_name()`, `get_public_suffix()` and `get_chosen_prefix() methods`.

The `ip_mappings` contains the list of IPs resolved by the domain, saved as `Python strings` objects.

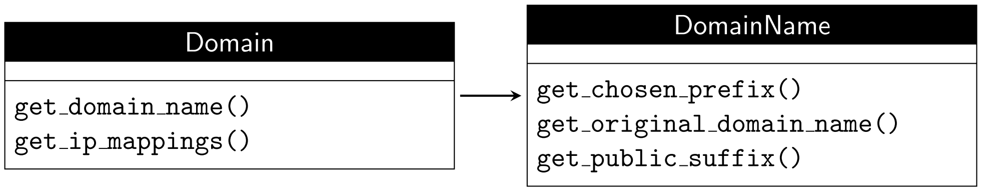


Figure 18: Domain and DomainName classes.

5.2.3 The Bootstrap Phase

The **Bootstap Phase** is reported in Listing 5.2. Cerberus first tries to load the ground truth from a saved file (lines 6–8). If this file is not found (i.e., the system was never bootstrapped before), Cerberus generates the list of AGDs clusters and saves it to `bootstrap.pk` using the `cPickle` module from the `Python` standard library, that offers functions to save objects into binary files. This technique allows Cerberus to be bootstrapped only once and then load the knowledge. We have implemented the possibility of manually issuing the `bootstrap` method, to ensure the capability of re-initializing the system with different blacklists.

Listing 5.2: bootstrap function implementation.

```
  def bootstrap(self):
          """Generates botnet clusters to classify unseen domains."""
3         logger.info("Bootstrapping Cerberus...")

          try:
6             with open('bootstrap.pk', 'rb') as f:
```

```
                self.dga_subclusters = pickle.load(f)
                logger.info("Cerberus bootstrap retrieved from file.")
        except IOError:
            bootstrap_cluster_factory = DomainClusterDatabaseFactory(
                identifier='exposure',
                experiment=False)
            bootstrap_cluster = bootstrap_cluster_factory.get()
            family_clusterer = FamilyClusterer(bootstrap_cluster)
            self.dga_subclusters = family_clusterer.compute_clusters()
            with open('bootstrap.pk', 'wb') as f:
                pickle.dump(self.dga_subclusters, f)
            logger.info("Finished bootstrapping Cerberus.")

        dispatcher.send(
            signal=signals.BOOTSTRAP_DONE,
            sender=self.__class__.__name__)
```

We would like to remember that the **Bootstrap Phase** *i)* produces knowledge in an unsupervised and automatic fashion and *ii)* it is not necessary to Cerberus to function, as the system is able to operate in a complete unsupervised way. If no knowledge is available when Cerberus is started for the first time, all the analyzed domains will be stored in the `suspicious_ips` database schema, and used by the **Time Detective** to extract the clusters (see Figure 16).

5.2.4 The Filtering Phase

The **Filtering Phase** is achieved by the `stream_filter.py` module, where we find the `sift_dns_stream` function. The `sift_dns_stream` function receives as input a list of filters, a data source and a synchronized queue, as implemented in the `multiprocessing` `Python` library, where to store the results. All the filters in the list must be an implementation of the abstract class `DNSStreamFilter`. This class requires all of its children to implement the `filter` method. In this way, the `sift_dns_stream` method can iterate on a list of filters and call the `filter` method on each one of them (see Figure 19). While the filters are applied in a serial fashion, each filter sifts the data in parallel.

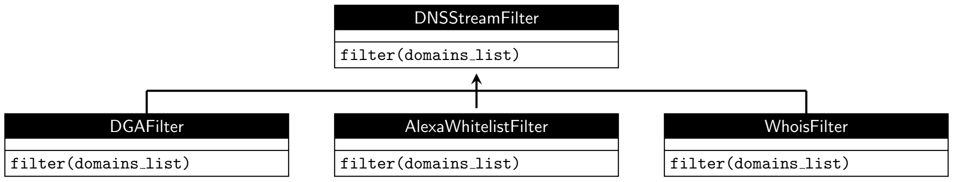


Figure 19: The abstract class DNSStreamFilter and three of its implementations.

The data is first sliced into N parts, where N is the number of cores automatically detected. Then N new processes are instantiated to process the slices. The domains that are not filtered out are stored in a shared and synchronized queue that at the end of the process is transformed into a list and returned. In other words we have applied the concepts of *map* and *reduce*, which perfectly fit our situation, where we have to apply the same operation (the `filter` method) to a large amount of independent data (the domains).

We designed the **Filtering Phase** having in mind the possibility of getting data from multiple sources. To this end, the `sift_dns_stream` function is not issued directly, but by the `sift_dns_streams` function. We provide this method with a list of `jobs`, objects of the `Job` class, which is initialized with a list of filters and a list of domains, thus offering the possibility of having multiple *jobs*, one for each source.

Listing 5.3: sift_dns_stream function implementation.

```python
def sift_dns_stream(filters, data_source, queue):
    """It sifts a single datasource. Datasource must be a list of Domain
    objects."""

    temp = data_source
    n_of_cores = multiprocessing.cpu_count()
    logging.info("Starting data analysys for %d documents", len(temp))

    for stream_filter in filters:
        if not isinstance(stream_filter, DNSStreamFilter):
            logging.error("Not a DNSStreamFilter object.)))
        else:
            logging.info("Filtering using " + stream_filter.name)
            temp_q = manager.Queue()
            if len(temp) == 0:
                continue
            step = len(temp) / min(n_of_cores, len(temp))
            slices = [temp[x:x+step] for x in xrange(0, len(temp), step)]
            processes = list()

            for i, s in enumerate(slices):
                p = Process(
                    target=sift_dns_stream_slice,
                    args=(s, temp_q, stream_filter,)
                )
                p.start()
                processes.append(p)

            for i, process in enumerate(processes):
                process.join()

            temp = list(chain.from_iterable(dump_queue(temp_q)))
            logging.info("Number of domains: " + str(len(temp)))

    queue.put(temp)
```

5.2.5 The Detection Phase

The **Detection Phase** includes **Classification** and **Clustering** of the unseen domains. In the next paragraphs we will go through the most important aspects concerning the implementation of these two tasks.

Classification

The **Classification** is performed by the `SVMSSKClassifier` class. Each instance is initialized *i)* with the clusters that share the IP address with the domain to be classified, and *ii)* the parameters to be used with the SSK, which are the decay factor λ and the substring length u. This class exposes two public methods, `train()` and `classify()`. The first one is used to train the SVM on the clusters it was initialized with. The SVM and the kernel functions are those implemented in the `Shogun` library for machine learning. The `classify()` method takes as input the domain to be classified and returns the identifier of the cluster the domain was labeled with.

Clustering

The clustering module `ssk_clustering.py` contains the classes involved in the clustering process. The `DBSCANSSKClusterer` class was implemented as a first abstraction layer to cluster together `strings` objects using the DBSCAN algorithm. It is initialized with the list of strings that form the initial population, then it computes the distance matrix using again the Subsequence String Kernel as implemented in the `Shogun` library. Then it leverages the DBSCAN algorithm from the `scikit-learn` library, which allows a precomputed distance to be used as a metric.

To analyze the DNS data we implemented the `DBSCANSSKDomainsClusterer`, subclass of `DBSCANSSKClusterer`, which accepts a list of `Domain` objects as input. We made this choice to respect the software engineering guidelines of modularity and extensibility. In fact in this way it is easy to deal with a change in the representation of the DNS data. Whatever the new class that replaces `Domain` can be, it is sufficient to implement just the translation to `string` objects.

In order to compare two clusters for deciding whether to merge them together or not, we implemented the `ClustersComparator` class. It is initialized with two clusters of domains, instances of the `DomainCluster` class implemented by Schiavoni et al. [22] in PHOENIX. `ClustersComparator` objects offer the `clusters_should_merge()` methods, which tells whether the clusters should merge by computing the Welch's test as implemented in the `scipy` library.

5.3 Summary

In this chapter we have presented various aspects concerning the implementation of CERBERUS. We summarize in Table V the technologies employed to tackle the different parts.

TABLE V: TECHNOLOGIES USED IN CERBERUS.

Aspect	Technology	Version
Programming Language	`Python`	2.7.3
Database System	`Mongo DB`	2.4.9
Calculus	`numpy`	1.7.1
Data Mining	`Shogun`	3.2.0
Data Mining	`scikit-learn`	0.14.1
Data Mining	`scipy`	0.12.0

In the next chapter we shall validate the goodness of CERBERUS.

CHAPTER 6

EXPERIMENTAL VALIDATION

In this chapter we describe the results that validated the effectiveness of CERBERUS. We had two goals in mind: first we wanted to confirm the effectiveness of the **Classifier**, which is easily quantifiable by terms of the confusion matrix produced; second, we wanted to test CERBERUS's goodness when deployed in the wild. To this end we let the system analyze one week of real DNS passive data and see if it *classifies* unseen domains belonging to known threats and if it *detects* new threats. This latter test is far from trivial: Given the massive quantitative of data it is very hard even to give a rough estimation of false negatives, as it would require to manually check every domain discarded in the **Filtering Phase**. Moreover some results, though important, cannot be quantified: For instance CERBERUS was able to correctly separate clusters of domains belonging to the same threat, but using different DGAs, and to merge together two clusters of domains employed by `Palevo`, brilliantly understanding that they belonged to the same botnet. Both of the aforementioned test are presented in the following of this chapter, highlighting the goodness of the obtained results.

Chapter Organization

The remainder of the chapter is organized in the following fashion:

- in Section 6.1 we precisely set our goals, what we want to prove with our experiments;
- in Section 6.2 we describe the dataset employed in our experiments;

- in Section 6.3 we test the effectiveness of the classifier;
- in Section 6.4 we run a one week simulation to see how the system would behave once deployed.

6.1 Goals

As discussed in Section 3.2.8, Phoenix suffers from two main kind of shortcomings, one *conceptual* whereas the other concerns the validation. The conceptual shortcomings regard the use of *ad hoc* parameters when it comes to classifying unseen domains, an approach that is prone to overfitting and, in our case, it is not able to correctly classify AGDs that feature a different TLD or a domain length that evades the previous thresholds. With our first experiment we want to validate our classifier: This test is described in Section 6.3. The second kind of shortcoming has to do with validation. Phoenix was not tested in the wild, whereas this is a mandatory step for a detection system that wants to be deployed in the real world. In Cerberus we have addressed this issue, and in Section 6.4 we confirm this claim.

6.2 Dataset

Cerberus' tests employed real passive DNS data collected in the wild. With the term *passive*, we refer to the technique invented by Weimer [28] in 2004, called "Passive DNS Replication", to obtain Domain Name System data from production networks, and store it in a database for later reference [28]. The sensors, deployed as close as possible to large caching servers, collect the data and send it to an *analyzer* module, where the data is filtered. After the filtering process, the data is transformed in a format that makes the querying process easier, and stored in a database. We were able to get access to approximately three months of data, from the 12th of January to the 15th of March 2013. About 609 M of DNS queries/replies were collected by the ISC/SIE monitor. Data statistics are summarized in Table VI. The data was divided

TABLE VI: ISC/SIE DATA SUMMARY STATISTICS.

Begin of Recording	Sat, 12 Jan 2013 18:19:57 GMT
End of Recording	Fri, 15 Mar 2013 23:37:11 GMT
Total Records	608,958,044

into snapshots of twenty minutes on average, counting 200,000 DNS messages, of which about 50,000 were successful (i.e., non `NXDOMAIN`) DNS replies.

6.3 The Classifier

We tested Cerberus's classifier against the ground truth generated by Phoenix, i.e., the clusters generated during the **Bootstrap Phase**. We employed data *automatically* labeled by Cerberus and not data *manually* labeled by humans as we wanted to run a test as close as possible to the real case scenario. Cerberus in fact, will automatically produce the labeled records later to be used for classification, and we wanted to have our experiment set up to reproduce such situation. Note that, though automatically labeled, the clusters' maliciousness and quality were manually assessed by Schiavoni et al. [22] and therefore represent a valid dataset to run our experiment.

6.3.1 Accuracy

We considered four clusters that counted 1,100 samples, thus we could measure the accuracy depending on the number of domains used up to 1,000 samples, and leave the remaining 100 domains for testing. We validated the classifier using *repeated random sub-sampling validation*, ten times for each amount of points. This means that, for instance, for ten times we randomly selected 200 points to train the classifier and 100 points to test it: This validation method allows to reveal the effects of overfitting, if any. We collected the overall accuracies from the confusion matrix along with the computation time. We repeated this operation until we counted 1,000 points in the training set, step 100. This data is reported in Table VII. Overall accuracy

TABLE VII: CERBERUS CLASSIFIER ACCURACY STATISTICS.

	Accuracy				
Domains	Avg	Std	Min	Max	Time (s)
1,000	0.927	$1.17 \cdot 10^{-2}$	0.905	0.950	101.49
900	0.926	$1.41 \cdot 10^{-2}$	0.892	0.948	86.56
800	0.928	$1.33 \cdot 10^{-2}$	0.907	0.948	73.59
700	0.916	$1.27 \cdot 10^{-2}$	0.897	0.940	62.72
600	0.920	$1.21 \cdot 10^{-2}$	0.895	0.940	50.88
500	0.917	$1.03 \cdot 10^{-2}$	0.897	0.933	42.44
400	0.918	$1.44 \cdot 10^{-2}$	0.887	0.938	29.24
300	0.910	$1.74 \cdot 10^{-2}$	0.880	0.933	21.37
200	0.895	$1.37 \cdot 10^{-2}$	0.877	0.920	14.13

grows and then stabilizes at about 93% from 800 points on (see Figure 20). For this reason the implementation of the **Classifier** has a sampling upper bound of 800 points to be randomly selected from the clusters to train the SVM used for classification.

6.3.2 Analysis of Classification Errors

We report in Figure 21 one of the confusion matrices, obtained using 1,000 samples for training, during the validation. As you can see in the picture the best performances are obtained

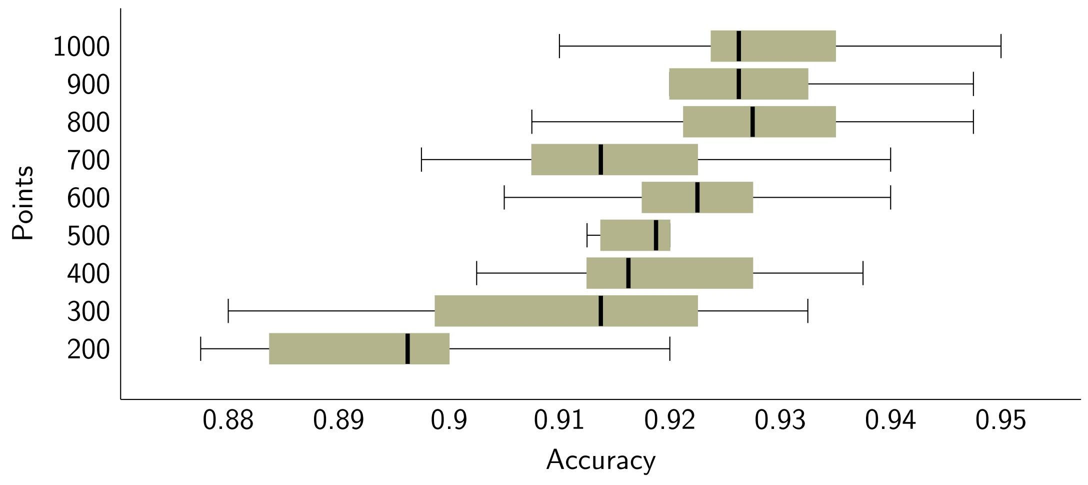


Figure 20: Cerberus classifier accuracies.

with cluster a, while the worst with cluster d. This holds for all the sub-sampling validations. This is due to the different lengths and distribution of characters that characterize the clusters. Cluster a in fact, is composed of domains that, on average, count more than thirty characters and are produced using a DGA that leverages a common hashing function over the date. This kind of algorithm generates domains that share a sort of "pattern" which allows the classifier to better recognize such domains. On the other hand, cluster d's domains are three characters long: Therefore it is harder to find "patterns" shared by the domains and consequently harder for the **Classifier** to classify the domains correctly. This is confirmed by looking at the distribution of the distances among the domains in the clusters, computed using the Subsequence String Kernel and reported in Figure 22. It is evident that distances in cluster a have a normal distribution with a very low mean and variance, which indicate very high intra-cluster similarity, whereas cluster `d`'s distances are randomly distributed and exhibit higher values, which indicate low intra-cluster similarity. Note that despite of this weakness, the overall accuracy allows us to use this classifier when Cerberus is deployed in the wild.

		Predicted			
		a	b	c	d
Actual	a	100	0	0	0
	b	1	92	6	1
	c	2	0	98	0
	d	3	0	6	91

a
`caaa89e...d4ca925b3e2.co.cc`
`f1e01ac...51b64079d86.co.cc`
b
`kdnvfyc.biz`
`wapzzwvpwq.info`
c
`jhhfghf7.tk`
`faukiijjj25.tk`
d
`cvq.com`
`epu.org`

Figure 21: Cerberus SSK classifier confusion matrix.

6.3.3 Training Speed

Training time grows, in a linear fashion, from 14.13 to 101.49 seconds (see Figure 23). As Cerberus is designed to analyze a *live* stream of DNS data, the training time is not negligible, as it could make the classification process too long. To address this issue we can store the trained SVM machines: For instance if an unseen domain d can belong either to cluster α or cluster β, Cerberus trains the SVM using cluster α and cluster β. Then this machine is stored

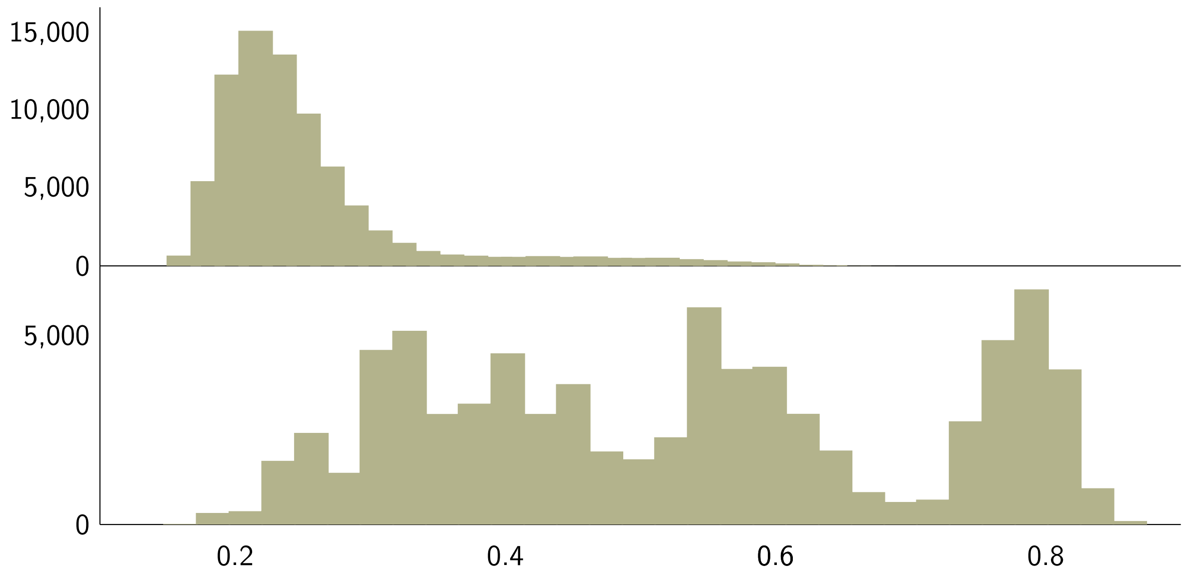


Figure 22: From the top: cluster `a`'s and cluster `d`'s distances distributions.

as a binary file using the `cPickle` `Python` library: When another unseen domain g can belong either to cluster α or cluster β, Cerberus retrieves the stored trained SVM, loads it in memory and uses it to label g, thus drastically improving the performances.

6.4 Cerberus in the Wild

This is the main test to prove the effectiveness of Cerberus. We decided to run a batch experiment of one week and one day of deployment in the wild, using real data from the aforementioned dataset (see Section 6.2). During the first week Cerberus leveraged the ground truth generated in the **Bootstrap Phase** to classify those unseen domains that would share their IP address with the clusters in the ground truth. Those unseen domains that would not share the IP address with any of the clusters were considered "suspicious", as were not

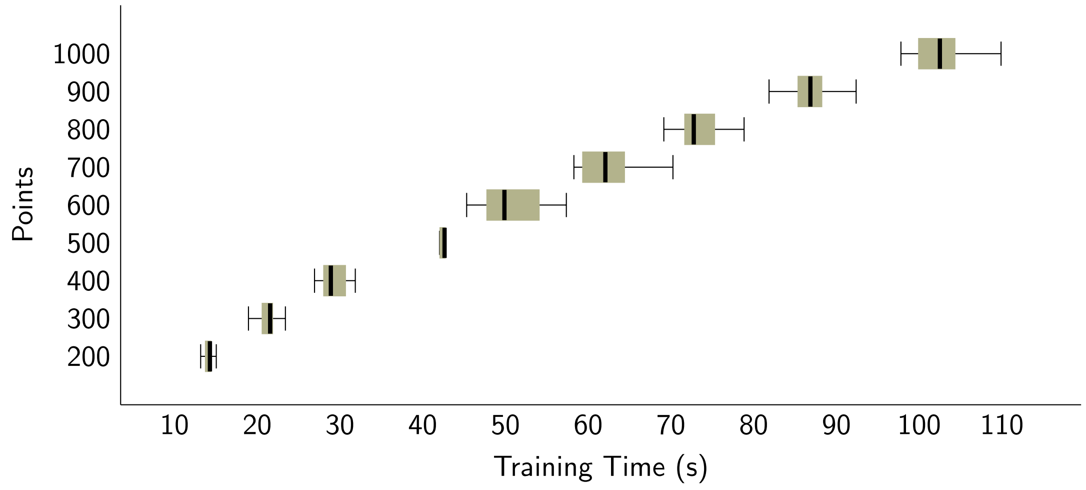


Figure 23: Cerberus classifier training time.

discarded by the **Filtering Phase** (i.e., they are likely-malicious), and stored in a database. Then, after one week time passed, the **Time Detective** ran the clustering routine on these "suspicious" domains: New clusters were found and added to the ground truth. The day after Cerberus used that knowledge to successfully label unseen domains belonging to previously unknown threats, drastically augmenting the number of detected malicious domains.

6.4.1 The Bootstrap

Before starting classifying data, Cerberus can be bootstrapped: If this happens the system can leverage the knowledge obtained to classify unseen domains. Otherwise Cerberus starts to function with no knowledge and will build its ground truth throughout time in an automatic fashion. We decided to bootstrap the system, in order to see how Cerberus behaves *before* and *after* it increases its knowledge, and to use the blacklist provided by Exposure [5], thus providing CERBERUS with the same knowledge used to feed Phoenix [22]. Once **The Bootstrap** is completed, CERBERUS possesses a list of eleven clusters of malicious domains likely generated by the same DGA: Among others we find clusters that Schiavoni et al. [22] confirmed referring to `Palevo` and `Conficker`. Two of the eleven clusters generated by PHOENIX in **Bootstrap Phase** are reported in Table VIII.

TABLE VIII: TWO OF THE ELEVEN CLUSTERS PRODUCED BY PHOENIX.

CLUSTER f105c		CLUSTER 0f468	
Threat:	Palevo	Threat:	Sality
IPs:	`176.74.176.175`	IPs:	`217.119.57.22`
	`208.87.35.107`		`91.215.158.57`
			`178.162.164.24`
			`94.103.151.195`
Domains:	`cvq.com`	Domains:	`jhhfghf7.tk`
	`epu.org`		`faukiijjj25.tk`
	`bwn.org`		`pvgvy.tk`

6.4.2 Collecting Data

The test started the day February, 7th 2013. At the end of the week, February, 14th 2013, roughly 22,000,000 DNS replies were analyzed, 187 domains were classified as malicious and labeled using the ground truth provided by PHOENIX in the **Bootstrap Phase** (see Table IX). We searched VirusTotal for the labeled domains and we were able to prove that 167 domains belonged to the `Conficker` botnet, while the remainder to other botnets or generic malware, including the `Flashback` botnet. Moreover 3,576 domains considered by Cerberus "suspicious" (i.e., they were not filtered out by the **Filtering Phase**) were stored together with their IP addresses: We counted exactly 1,300 distinct IP addresses, which means that multiple "suspicious" domains resolved to the same IP.

TABLE IX: A SAMPLE OF THE MALICIOUS DOMAINS CLASSIFIED DURING THE FIRST WEEK.

Labeled 07e21		
Threat:	Conficker	
Domains:	`hhdboqazof.biz`	`tnoucgrje.biz`
	`poxqmrfj.biz`	`gwizoxej.biz`
	`hcsddszzzc.ws`	`jnmuoiki.biz`

6.4.3 Producing New Knowledge

At the end of the week Cerberus performed the clustering routine over the domains resolving to suspicious IP addresses. As described in Section 4.2.3, the clustering routine is performed only after grouping the IP addresses by Autonomous System. We report in Table X the main ASs involved in the analysis. We applied the clustering routine, which yielded 47 clusters, of which the bigger ones (counting more than 25 elements) are reported in Table XI along with the related threat. This means that, taking as input only the passive DNS traffic, Cerberus was able to identify, in a completely automatic fashion, groups of IPs that are associated to

TABLE X: AUTONOMOUS SYSTEMS OF THE DOMAINS IN CLUSTERING PHASE.

AS	Network Name	Country
15456	INTERNETX-AS InterNetX GmbH	DE
22489	Castle Access Inc	UK
47846	Sedo GmbH	DE
53665	BODIS-1 - Bodis, LLC	CN

TABLE XI: CERBERUS' NEW CLUSTERS (ASTERISKS TO MATCH TABLE ??).

Threat	AS	IPs	Size
Sality	15456	`62.116.181.25`	26
Palevo	53665	`199.59.243.118`	40
Jadtre*	22489	`69.43.161.180`	173
		`69.43.161.174`	
Jadtre**	22489	`69.43.161.180`	37
Jadtre***	22489	`69.43.161.167`	47
Hiloti	22489	`69.43.161.167`	24
Palevo	47846	`82.98.86.171`	142
		`82.98.86.176`	
		`82.98.86.175`	
		`82.98.86.167`	
		`82.98.86.168`	
		`82.98.86.165`	
Jusabli	30069	`69.58.188.49`	73
Generic Trojan	12306	`82.98.86.169`	57
		`82.98.86.162`	
		`82.98.86.178`	
		`82.98.86.163`	

TABLE XII: JADTRE THREATS SAMPLE DOMAINS.

Cluster	IP	Sample Domains	
Jadtre*	69.43.161.180	379.ns4000wip.com	78.ns4000wip.com
	69.43.161.174	418.ns4000wip.com	272.ns4000wip.com
		285.ns4000wip.com	98.ns4000wip.com
Jadtre**	69.43.161.180	391.wap517.net	137.wap517.net
		251.wap517.net	203.wap517.net
		340.wap517.net	128.wap517.net
Jadtre***	69.43.161.167	388.ns768.com	312.ns768.com
		353.ns768.com	153.ns768.com
		296.ns768.com	30.ns768.com

malicious botnet activity (e.g., C&C). This is new knowledge, which an investigator can use to find new botnet servers. There are three clusters, two of which sharing the IP address `69.43.161.180` and all of three residing in the same AS, that are labeled with the same threat, `Jadtre`[1]. The reason why they are not one cluster, though belonging to the same threat, is because they were generated using three different DGAs, as it is clear from Table XII. This proves that CERBERUS is able of a fine grained detection of malicious activities, showing its capability of isolating not only different threats, but different DGA used by the same threat. The new clusters were then added to the ground truth. CERBERUS ran a *similarity check* to see whether the new clusters should be merged together with the old ones. The Welch's test told that there was not enough statistical evidence to consider clusters a and b, reported in Table XIII,

[1]`http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=TrojanDownloader:Win32/Jadtre.A`

dissimilar, thus Cerberus decided to merge them together. Further investigations[1] confirmed that the IP addresses from both cluster *a* and cluster *b* belonged to `Palevo` C&Cs. This means that Cerberus was able to understand that the domains of the two clusters, the first generated by Phoenix from the Exposure blacklist, and the second discovered by Cerberus analyzing real passive DNS data, were produced by the same DGA. This discovery lead the ground truth to be enriched, as the IP address `199.59.243.118` was added to the cluster, together with the domains. This means that Cerberus was able to successfully discover a *migration* by leveraging only the linguistic features computed by the SSK. After the new clusters were added to

TABLE XIII: CLUSTERS MERGED BY CERBERUS.

Cluster a		Cluster b	
IPs:	`176.74.176.175`	IPs:	`82.98.86.171`
	`208.87.35.107`		`82.98.86.176`
			`82.98.86.175`
			`82.98.86.167`
			`82.98.86.168`
			`82.98.86.165`
Sample Domains	`cvq.com`	Sample Domains	`knw.info`
	`epu.org`		`rrg.info`
	`bwn.org`		`nhy.org`
	`lxx.net`		`ydt.info`

[1]`https://palevotracker.abuse.ch/`

the ground truth, Cerberus started again the **Detection Phase**, leveraging the increased knowledge: The next day Cerberus classified 319 malicious domains, while during the whole previous week counted 187 malicious domains, on average 26 domains a day. Hence, Cerberus was able to increase its knowledge in a completely automatic and unsupervised fashion and use this enhanced knowledge to drastically (twelve times as much) augment the number of daily classified malicious domains.

6.4.4 Summary

Firstly Cerberus was bootstrapped using the Exposure [5] blacklist, extracting eleven clusters of domains referring to the same DGA. Then for one week Cerberus analyzed a stream of passive DNS data collected in the wild by a ISC/SIE DNS monitor. During that week 187 malicious domains were detected and 1,300 IP addresses were labeled as "suspicious". At the end of the week the clustering routine produced new knowledge in the form of 47 new clusters, which were added to the ground truth. One of them was automatically merged by Cerberus and a check of the IP addresses on the Web confirmed that both clusters belong to the `Palevo` botnet. Therefore Cerberus was able to

1. perform on-line detection of known threats;
2. automatically detect new threats;
3. increase its knowledge;
4. use the new knowledge to classify previously unknown threats.

We think that these results are quite encouraging and so prove the effectiveness of Cerberus. Obviously there is more that can be done and there are some difficulties to overcome. These matters shall be addressed in the next chapter.

CHAPTER 7

CONCLUSIONS

Cerberus results seem promising and it is therefore worth continuing to follow this approach. In this chapter we first address the *limitations* CERBERUS suffers from, mainly related to the **Detection Phase**. Then we will discuss the possible further developments that could fix the aforementioned issues and enhance the system. Finally we will try to briefly summarize what we achieved with CERBERUS.

Chapter Organization

The remainder of the chapter is organized in the following fashion:

- in Section 7.1 we discuss the current limitations of CERBERUS;
- in Section 7.2 we discuss how to possibly address these limitations and how to enhance CERBERUS presenting a few further developments.

7.1 Limitations

Cerberus suffers from two main limitations, both related to the **Detection Phase**. The first one is encountered when an unseen domain d does not belong to any of the clusters the SVM was trained on (i.e. the clusters sharing their IP address with d): Cerberus will anyway assign it to the most "similar" cluster. This is an issue that needs to be solved. We need to find a test that allows us to tell when a single domain does not belong to any of the clusters selected to train the SVM. If this is the case we could design a temporary buffer where these domains are stored, and perform the clustering routine on this buffer every γ time to isolate new clusters. This situation is depicted Figure 24: Figure 24 (a) reports sample domains from

Cluster 3e774	
`16542.com`	`79581.com`
`44962.com`	`46096.com`
`72879.com`	`09941.com`
`42661.com`	`90338.com`
`47176.com`	`69313.com`
`75827.com`	`34171.com`

(a)

Labeled 3e774
`asdfuh982hdodjc.com`
`open-932978.com`
`hp21821867626.mygateway.net`
`www2.h3xa.com`
`061107dd0208.agulhal.com`
`clkh71yhks66.com`

(b)

Figure 24: Samples from cluster 3e774 and misclassified unseen domains.

the cluster 3e774, generated during the **Bootstrap Phase**. These domains share an easily identifiable pattern: They all count five digits and exhibit the `.com` TLD. In Figure 24 (b) there are a few domains labeled as 3e774, while it is clear that they were not produced using the same DGA.

The second major limitation regards the clustering routine. It could be the case that the attacker's servers span across more than one Autonomous System: in our test in the wild there was just one recorded case, still it is not a phenomenon to be ignored. E.g., in Table XIV is reported a cluster of domains found by clustering on the flattened list of domains related to the suspicious IPs instead of grouping them by Autonomous System. We did not follow this approach in Cerberus because even though it is possible to find a minor number of clusters otherwise untraceable, the overall clustering quality is seriously worsened.

TABLE XIV: CLUSTER FOUND WITHOUT GROUPING BY AS.

Domains	IP
j5.kwu.rcvn.biz	64.191.8.11
1a.flw.rcvn.biz	184.22.158.49
vn.oxw.rcvn.biz	173.212.224.158
sp.ebx.rcvn.biz	37.1.217.136
nm.iwk.rcvn.biz	64.191.8.32
41.ajs.rcvn.biz	184.82.98.22
a3.fxw.rcvn.biz	64.191.58.73
70.axw.rcvn.biz	96.9.139.111
df.wek.rcvn.biz	66.96.248.248

7.2 Future Works

We start from the limitations listed in the previous section to elicit the future developments of Cerberus. The first and crucial improvement to be done is to find a way to determine whether a single domain is coherent with the cluster it is about to be put into. This is crucial because, as stated in the previous section, an IP address could serve more than one malicious activity, each one possibly employing a different DGA. The second crucial development concerns the clustering routine, as Cerberus is not able of detecting malicious activities that span different Autonomous Systems. One possible way to address this issue is to combine the clustering results obtained when grouping and when not grouping by AS.

One improvement regards the Classifier. Shafer and Vovk [23] proposed the *Conformal Predictor*, a tool that tells you how much confident you can be about a prediction, based on the history of your past predictions. We would like this component to be added to Cerberus, as we would have a way to determine how much confident we can be when labeling an unseen domain to Cerberus's ground truth.

One important future development is to save the SVMs as they are trained, storing a *dictionary* where the keys are the clusters the SVM was trained on and the values are the SVMs objects themselves.

Finally we would like to deploy Cerberus and analyze real time data. Then we could develop a web application where the user can *i)* ask the system whether a domain or an IP is malicious (i.e., that domain or IP is in one of the clusters of the ground truth), *ii)* download the ground truth and *iii)* upload a dump of traffic (think of a corporate network) and have Cerberus analyzing it to *detect* and *discover* malicious activities.

7.3 Concluding Remarks

We presented Cerberus, a system able to detect and characterize unseen DGA-based threats, botnets above all, easily deployable and free of privacy related issues as it analyzes passive DNS data, and able to operate without any *a priori* knowledge. We started from and improved the results obtained by Phoenix [22], being able to discover new threats that rely on C&C servers that exhibit unseen IP addresses, and building a better classifier which is now trained not on *ad hoc* parameters, but leverages the domain name itself as classification feature.

We tested Cerberus in the wild, analyzing one week of passive DNS data collected from the 7th to the 14th of February, 2013. The system, without any *a priori* knowledge, was able to isolate 47 clusters of malicious domains belonging to various threats among which we find the well known `Palevo`, `Sality` and `Jadtre`. Moreover we tested the classifier in a four classes classification test, featuring an overall precision of 93% with 800 points in the training set.

Therefore, given the results obtained, we argue that Cerberus improves the systems proposed so far by adopting an *unsupervised* approach and analyzing passive DNS data, two features that allow Cerberus to automatically detect previously unseen DGA-based threats in the Internet.

CITED LITERATURE

[1] Alpaydin, E. *Introduction to machine learning.* MIT press: 2004. [Citation at page 69]

[2] Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou, II, N., Dagon, D.: Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 27–27, Berkeley, CA, USA, 2011. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=2028067.2028094`. Retrieved on April 21, 2014. [Citations at pages 2, 42, 43, 44 and 55]

[3] Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: Detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=2362793.2362817`. Retrieved on April 21, 2014. [Citations at pages 23, 43, 48, 49, 55 and 56]

[4] Bailey, M., Cooke, E., Jahanian, F., Xu, Y., Karir, M.: A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 299–304, March 2009. doi: 10.1109/CATCH.2009.40. [Citations at pages 2 and 11]

[5] Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive DNS analysis. In *Proceedings of NDSS*, 2011. [Citations at pages 43, 44, 45, 47, 66, 104 and 111]

[6] Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., Kruegel, C.: Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 129–138, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1312-4. doi: 10.1145/2420950.2420969. URL `http://doi.acm.org/10.1145/2420950.2420969`. Retrieved on April 21, 2014. [Citations at pages 2, 43, 46 and 55]

[7] ENISA: ENISA Threat Landscape, Responding to the Evolving Threat Environment, 2012. [Citation at page 1]

[8] ENISA: ENISA Threat Landscape, Mid-year 2013, 2013. [Citations at pages 1 and 11]

[9] Feily, M., Shahrestani, A., Ramadass, S.: A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 268–273, June 2009. doi: 10.1109/SECURWARE.2009.48. [Citations at pages 8 and 16]

[10] Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C.J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A., et al.: Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 821–832. ACM, 2012. [Citation at page 1]

[11] Haddadi, F., Zincir-Heywood, A.: Analyzing string format-based classifiers for botnet detection: Gp and svm. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2626–2633, June 2013. doi: 10.1109/CEC.2013.6557886. [Citations at pages 50, 51, 52 and 55]

[12] Haddadi, F., Kayacik, H.G., Zincir-Heywood, A.N., Heywood, M.I.: Malicious automatically generated domain name detection using stateful-sbb. In *Applications of Evolutionary Computation*, pages 529–539. Springer, 2013. [Citations at pages 43, 50, 69 and 70]

[13] Holz, T., Gorecki, C., Rieck, K., Freiling, F.C.: Measuring and Detecting Fast-Flux Service Networks. In *Symposium on Network and Distributed System Security*, 2008. [Citation at page 66]

[14] Hubert, L., Schultz, J.: Quadratic assignment as a general data analysis strategy. *British Journal of Mathematical and Statistical Psychology*, 29(2):190–241, 1976. [Citation at page 75]

[15] Jarvis, K.: CryptLocker Ransomware, 2013. URL `http://www.secureworks.com/cyber-threat-intelligence/threats/cryptolocker-ransomware/`. Retrieved on April 21, 2014. [Citations at pages 30, 31, 34 and 35]

[16] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002. ISSN 1532-4435. doi: 10.1162/153244302760200687. URL `http://dx.doi.org/10.1162/153244302760200687`. Retrieved on April 21, 2014. [Citations at pages 4, 51, 70 and 71]

[17] McAfee Labs: McAfee Threats Report: First Quarter 2013, 2013. URL `http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2013.pdf`. Retrieved on April 21, 2014. [Citations at pages 1 and 29]

[18] Neugschwandtner, M., Comparetti, P.M., Platzer, C.: Detecting Malware's Failover C&C Strategies with Squeeze. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 21–30, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0672-0. doi: 10.1145/2076732.2076736. URL `http://doi.acm.org/10.1145/2076732.2076736`. Retrieved on April 21, 2014. [Citation at page 2]

[19] Perdisci, R., Corona, I., Giacinto, G.: Early detection of malicious flux networks via large-scale passive DNS analysis. *Dependable and Secure Computing, IEEE Transactionson*, 9 (5):714–726, 2012. [Citations at pages 43, 47, 48 and 55]

[20] Rossow, C., Dietrich, C.J., Bos, H., Cavallaro, L., van Steen, M., Freiling, F.C., Pohlmann, N.: Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, BADGERS '11, pages 78–88, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0768-0. doi: 10.1145/1978672.1978682. URL `http://doi.acm.org/10.1145/1978672.1978682`. Retrieved on April 21, 2014. [Citation at page 50]

[21] Sancho, D., Link, R.: Sinkholing Botnets, A Trend Micro Technical Paper, 2011. URL `http://www.trendmicro.co.uk/media/misc/sinkholing-botnets-technical-paper-en.pdf`. Retrieved on April 21, 2014. [Citations at pages 18 and 25]

[22] Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S.: Phoenix: DGA-based botnet tracking and intelligence. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, London, UK, 2014. Springer-Verlag.

[Citations at pages viii, 2, 3, 11, 16, 18, 23, 43, 47, 50, 52, 53, 54, 55, 56, 67, 68, 88, 95, 99, 105 and 116]

[23] Shafer, G., Vovk, V.: A tutorial on conformal prediction. *J. Mach. Learn. Res.*, 9:371–421, June 2008. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1390681.1390693`. Retrieved on April 21, 2014. [Citation at page 115]

[24] Sharifnya, R., Abadi, M.: A novel reputation system to detect dga-based botnets. In *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*, pages 417–423, Oct 2013. doi: 10.1109/ICCKE.2013.6682860. [Citations at pages 2, 43, 49, 50 and 55]

[25] Spagnuolo, M.: Bitiodine: Extracting intelligence from the bitcoin network. Master's thesis, Politecnico Di Milano, Piazza Leonardo da Vinci 32, Milan, December 2013. [Citations at pages 2, 9, 37 and 39]

[26] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., Vigna, G.: Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-894-0. doi: 10.1145/1653662.1653738. URL `http://doi.acm.org/10.1145/1653662.1653738`. Retrieved on April 21, 2014. [Citations at pages 2, 21, 25, 26, 27 and 28]

[27] Stone-Gross, B., Kruegel, C., Almeroth, K., Moser, A., Kirda, E.: Fire: Finding rogue networks. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 231–240. IEEE, 2009. [Citation at page 47]

[28] Weimer, F.: Passive dns replication. In *FIRST Conference on Computer Security Incident*, 2005. [Citation at page 98]

[29] Xu, W., Wang, X.: Think you know fast-flux domains? think again, 2013. URL `https://media.blackhat.com/us-13/US-13-Xu-New-Trends-in-FastFlux-Networks-Slides.pdf`. Retrieved on April 21, 2014. [Citation at page 66]

VITA

Edoardo Giovanni Colombo

EDUCATION

- Diploma di Liceo Scientifico (High School Diploma equivalent) at Liceo Scientifico Statale Leonardo da Vinci, Milano.

  Final grade: 100/100, year: 2008.

- Laurea (B.Sc. equivalent) in Ingegneria Informatica at Politecnico di Milano.

  Final grade: 103/110, year: 2011.

- Attending Laurea Magistrale (M.Sc. equivalent) in Ingegneria Informatica at Politecnico di Milano.

- Attending Master of Science in Computer Science at University of Illinois at Chicago.

  Final GPA: 4.0/4.0.