2012-ENST

EDITE - ED 130

Ph.D. ParisTech

D I S S E R T A T I O N

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy from

TELECOM ParisTech

Specialization « Internet and System's Security »

presented and defended by

Leyla BILGE

on the 15th of December 2011

Network Based Botnet Detection

Thesis supervisor : **Prof. Engin Kirda**

T
H
È
S
E

TELECOM ParisTech

école de l'Institut Télécom - membre de ParisTech

2012-ENST

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Sécurité d'Internet et des systèmes »

présentée et soutenue publiquement par

Leyla BILGE

le 15 Décembre 2011

La Détection des Botnet par l'Analyse de Réseau

Directeur de thèse : **Prof. Engin KIRDA**

TELECOM ParisTech

école de l'Institut Télécom - membre de ParisTech

T H È S E

Bubacığıma...

Abstract

The days when the Internet used to be an academic network with no malicious activity are long gone. Today, there is a high incentive for cyber-criminals to engage in malicious, profit-oriented illegal activity on the Internet. A popular tool of choice for digital criminals today are *bots*. Compared to other types of malware, the distinguishing characteristic of a bot is its ability to establish a command and control (C&C) channel that allows an attacker to remotely control or update a compromised machine. A number of bot-infected machines that are combined under the control of a single, malicious entity are referred to as a *botnet*. Such botnets are often abused as platforms to launch denial of service to send spam or to host scam pages.

In this thesis, we propose three network-based botnet detection techniques. Each technique models the detections by analyzing different types of network data : the first detection technique performs packet level inspection. The second one analyzes the DNS traffic to find the domains that are abused for different kinds of malicious purposes including being assigned for the command and control servers. And finally, the last one detects command and control servers by analyzing NetFlow data.

We propose a detection approach to identify single, bot-infected machines without any prior knowledge about command and control mechanisms or the way in which a bot propagates. Our detection model leverages the characteristic behavior of a bot, which is that it (a) receives commands from the botmaster, and (b) carries out some actions in response to these commands. The basic idea of our system is that we can generate detection models by observing the behavior of bots that are captured in the wild. Based on the observations of commands and responses, we generate detection models that can be deployed to scan network traffic for similar activity, indicating the fact that a machine is infected by a bot.

We introduce a passive DNS analysis approach and a detection system, EXPOSURE, to effectively and efficiently detect domain names that are involved in malicious activity. We use a set of features that allow us to characterize different properties of DNS names and the ways that they are queried. Our experiments with a large, real-world data set consisting of 100 billion DNS requests, and a real-life deployment for two weeks in an ISP show that our approach is scalable and that we are able to automatically identify unknown malicious domains that are misused in a variety of malicious activity (such as for botnet command and control, spamming, and phishing).

Finally, we present Disclosure, a large-scale, wide-area botnet detection system that incorporates a combination of novel techniques to overcome the challenges imposed by the use of NetFlow data. In particular, we identify several groups of features that allow Disclosure to reliably distinguish C&C channels from benign traffic using NetFlow records : (i) flow sizes, (ii) client access patterns, and (iii) temporal behavior. We demonstrate that these features are not only effective in detecting current C&C channels, but that these features are relatively robust against expected countermeasures future botnets might deploy against our system. Furthermore, these features are oblivious to the specific structure of known botnet C&C protocols.

Résumé

La période ou Internet était un réseau universitaire sans activités malveillantes est révolue depuis longtemps. De nos jours, les cyber-criminels ont beaucoup plus de raisons et de motivations pour conduire des activités illégales à but lucratif. Un des outils les plus reconnus pour les criminels numériques sont les bots. Un botnet correspond à un ensemble de plusieurs machines "infectées" qui sont sous le contrôle d'une seule entité malveillante. Ces botnets sont souvent utilisées comme des plateformes pour lancer des attaques de déni de service, pour envoyer des spams ou pour héberger des pages d'arnaques.

Cette thèse propose trois techniques de détection de botnet en se basant sur l'analyse du réseau. Chaque technique modélise ces détections en analysant différents types de données : la première technique effectue une ispection des données au niveau paquet. La deuxième technique analyse le traffic DNS pour retrouver des domaines qui sont affectés pour des fins malveillantes notamment par des serveurs de commandes et de contrôle. Enfin, la dernière technique détecte des serveurs de commande et de contrôle en analysant les données du flux réseau.

Table des matières

Table des figures

Liste des tableaux

Acronyms

These are the main acronyms used in this document. The meaning of an acronym is usually indicated once, when it first occurs in the text.

C&C	Command and Control
P2P	Peer-to-peer
DNS	Domain Name System
TTL	Time to Live
IRC	Internet Relay Chat
HTTP	Hypertext Transform Protocol
AUC	Area Under the ROC Curve
DT	Detection Rate
FP	False Positives Rate
LMS	Longest Meaningful Substring
SCADA	Supervisory Control and Data Acquisition
AV	Anti Virus
ISP	Internet Service Provider
DDoS	Distributed Denial of Service Attack
DGA	Domain Generation Algorithm
IDS	Intrusion Detection System
URL	Uniform Resource Locator
CPD	Change Point Detection Algorithm
SMTP	Simple Mail Transfer Protocol
AS	Autonomous System

Chapitre 1

Introduction

The days when the Internet used to be an academic network with no malicious activity are long gone. Today, the Internet has become a critical infrastructure, and it now plays a crucial role in communication, finance, commerce, and information retrieval. It has been reported that there are more than 2.7 billion web pages on the Internet now [108].

Unfortunately, as a technology becomes popular, it also attracts people with malicious intentions. In fact, digital crime is a growing challenge for law enforcement agencies. As Internet-based attacks are easy to launch and difficult to trace back, such crimes are not easy to prosecute and bring to justice. As a result, there is a high incentive for cyber-criminals to engage in malicious, profit-oriented illegal activity on the Internet. Regrettably, the number and sophistication of Internet-based attacks have been steadily increasing in the last ten years [97].

A popular tool of choice for digital criminals today are *bots*. A bot is a type of malware that is created with the intent of compromising and taking control of hosts on the Internet. It is typically installed on the victim's computer by either exploiting a software vulnerability in the web browser or the operating system or by using social engineering techniques to trick the victim into installing the bot herself. Compared to other types of malware, the distinguishing characteristic of a bot is its ability to establish a command and control (C&C) channel that allows an attacker to remotely control or update a compromised machine [40]. A number of bot-infected machines that are combined under the control of a single, malicious entity (called the *botmaster*) are referred to as a *botnet*. Such botnets are often abused as platforms to launch denial of service attacks [71], to send spam [54, 84], or to host scam pages [13]. Using a botnet, the attackers also typically steal sensitive information on a victim's machine (e.g., credit card numbers, chat logs, social network account credentials, etc.) [92].

Botnets have also been reported to have been used in attacks against nations – both intentionally, and by coincidence. For example, in 2007, there was a deliberate and organized bot-based distributed denial of service attack against the critical infrastructures of Estonia [41]. In 2009, Conficker infected the computers of three European armies. Hence, French fighter planes were prevented from taking off, the networks of the British and German army bases were partially shutdown, and a number of UK police forces had to disconnect their networks from the Internet [98]. Recently, the Stuxnet botnet attacked a critical infrastructure of a specific nation [74, 96]. In fact, Stuxnet appears to have been written specifically to attack a particular brand of Supervisory Control and Data Acquisition (SCADA) systems. SCADA systems are typically responsible for the operation of key components in power plants, pipelines, power distribution networks, and other similar industrial systems.

Traditional means of defense against bots rely on anti-virus (AV) software installed on end-users' machines as well as other type of malware. Unfortunately, as the existence of numerous botnets demonstrates, these systems are insufficient. The reason is that they rely on signatures of known samples, a well-documented limitation [32] that makes it difficult to keep up with the fast evolution of malware. To mitigate this limitation, a number of host-based defense systems have been introduced. These systems use static [33, 62] or dynamic [59, 111] code analysis techniques to capture the behavior of unknown programs. By comparing the observed behavior to a model that specifies characteristics of certain types of malware, previously unknown instances of malicious code can be identified. However, although useful, these systems are problematic in practice, as they incur a considerable runtime overhead and require each user to install the analysis platform.

To complement host-based analysis techniques, it is desirable to have a network-based detection system available that can monitor network traffic for indications of bot-infected machines. In practice, network-based detection systems are more preferable than host-based ones. One reason is that network-based detection systems have a complete visibility over the network while host-based techniques target only single individuals. Unfortunately, in the security ecosystems, users are usually the weakest link. Therefore, the network administrators often employ network-based detection systems that can be deployed to a vantage point in the network. In this way, they are able to monitor the network activities of all individuals without giving privelages to the computers that have already been protected by a host-based malware detection system.

Another reason is that the nature of botnets is prone to network-based detection systems. The characteristic feature of botnets is the command and control protocol they adopt. As the command and control infrustructure relies on a network communication, therefore observable in the network traffic, it is considered to be the tender spot of botnets. Even if botmasters apply some obfuscation techniques to hide the semantics of the command and control protocol, since it is transmitted over network, the packets remain in the network.

On these grounds, in this thesis we propose three network-based botnet detection techniques. Each technique models the detections by analyzing different types of network data : the first detection technique performs packet level inspection. The second one analyzes the DNS traffic to find the domains that are abused for different kinds of malicious purposes including being assigned for the command and control servers. And finally, the last one detects command and control servers by analyzing NetFlow data.

1.1 Botnet Detection Through Network Level Packet Inspection

Previous work to detect bots performing network-level packet inspection has proceeded along two main lines : The first line of research uses *vertical correlation* techniques. These techniques focus on the detection of individual bots, typically by checking for traffic patterns or content that reveal command and control traffic or malicious, bot-related activities. These systems require prior knowledge about the command and control channels and the propagation vectors of the bots that they can detect. For example, Rishi [49] analyzes IRC traffic for nicknames that are frequently used by bots, while the system proposed by Binkley and Singh [26] checks for suspicious IRC traffic statistics. BotHunter [52] is more advanced in that the tool combines alerts from both anomaly-based and signature-based intrusion detection systems to identify bot-related traffic. Nevertheless, as pointed out by the authors in a follow-up paper [51], BotHunter relies on the fact that the “bot behavior follows a *pre-defined* infection life cycle dialog model,” which is geared toward bots that use random scanning and well-known bot commands.

The second line of research to detect bots uses *horizontal correlation* approaches to analyze the network traffic for patterns that indicate that two or more hosts behave similarly. Such similar patterns are often the result of a command that is sent to several members of the same botnet, causing the bots to react in the same fashion (e.g., by starting to scan or to send spam). The salient property of techniques that use horizontal correlation, such as BotSniffer [53], BotMiner [51], and TAMD [86], is that they do not require *a priori* information about the way in which the command and control channel is implemented. The drawback of these approaches is that they cannot detect individual bots. That is, it is necessary that at least two hosts in the monitored network(s) are members of the *same* botnet. Given the general trend towards smaller botnets [36] and the possibility for a botmaster to assign two bots in the same network range to different botnets, this is a significant drawback.

The first botnet detection technique [109] presented in this thesis proposes a detection approach to identify single, bot-infected machines without any prior knowledge about command and control mechanisms or the way in which a bot propagates. Our detection model leverages the characteristic behavior of a bot, which is that it (a) receives commands from the botmaster, and (b) carries out some actions in response to these commands. Similar to previous work, we assume that the command and response activity results in some kind of network communication that can be observed.

The basic idea of our system is that we can generate detection models by observing the behavior of bots that are captured in the wild. More precisely, by launching a bot in a controlled environment and recording its network activity (*traces*), we can observe the commands that this bot receives as well as the corresponding responses. To this end, we present techniques that allow us to identify points in a network trace that likely correlate with response activity. Then, we analyze the traffic that precedes this response to find the corresponding command. Based on the observations of commands and responses, we generate detection models that can be deployed to scan network traffic for similar activity, indicating the fact that a machine is infected by a bot. Our approach produces specific detection models that are tailored to bot families or groups of bots related by a common C&C infrastructure. Because the system is automated, however, it is easy to quickly generate new models for bots that implement novel commands and responses. This is independent of any prior knowledge of the protocol or the commands that the bot uses.

1.2 Botnet Detection Through Passive DNS Analysis

Since the first malware appeared in the wild, there is an ongoing arms-race between malware authors and malware defense mechanisms. Each time a malware detection system was developed, malware modified itself to evade these systems. As a result, this arms-race caused the malware evolution.

Botnets are one another type of malware. They also developed techniques to thwart existing botnet detection mechanisms. As a first step, they applied encryption or obfuscation to hide the internals of their C&C infrastructure. Unfortunately, most of the botnet detection systems that perform network-level packet inspection, including our system that we described above, are limited in the fact that they cannot cope with botnets that obfuscate/encrypt their traffic. Therefore, the need for a new complementary botnet detection system is obvious.

One of the technical problems that attackers face when designing their malicious infrastructures is the question of how to implement a reliable and flexible server infrastructure, and command and control mechanism. Ironically, the attackers are faced with the same engineering challenges that global enterprises face that need to maintain a large, distributed and reliable service infrastructure for their customers. For example, in the case of botnets, the attackers need to efficiently manage remote hosts that may easily consist of thousands of compromised end-user machines. Obviously, if the IP address of the command and control server is hard-coded into the bot binary, there exists a single point of failure for the botnet. That is, from the point of view of the attacker, whenever this address is identified and is taken down, the botnet would be lost.

The Domain Name System (DNS) is a hierarchical naming system for computers, services, or any resource connected to the Internet. Clearly, as it helps Internet users locate resources such as web servers, mailing hosts, and other online services, DNS is one of the core and most important components of the Internet. Unfortunately, besides being used for obvious benign purposes, domain names are also popular for malicious use. For example, domain names are increasingly playing a role for the management of botnet command and control servers, download sites where malicious code is hosted, and phishing pages that aim to steal sensitive information from unsuspecting victims.

In order to better deal with the complexity of a large, distributed infrastructure, botnets have been increasingly making use of domain names. By using DNS, botmasters acquire the flexibility to change the IP address of the malicious servers that they manage. Furthermore, they can hide their critical servers behind proxy services (e.g., using Fast-Flux [100]) so that their malicious server is more difficult to identify and take down.

Using domain names gives botnet controllers the flexibility of migrating their servers with ease. That is, the botnet infrastructures become more "fault-tolerant" with respect to the IP addresses where they are hosted.

Our key insight is that as malicious services (e.g. botnets) are often as dependent on DNS services as benign services, being able to identify malicious domains as soon as they appear would significantly help mitigate many Internet threats that stem from botnets. Also, our premise is that when looking at large volumes of data, DNS requests for benign and malicious domains should exhibit enough differences in behavior that they can automatically be distinguished.

With our second botnet detection technique [25], we introduce a passive DNS analysis approach and a detection system, EXPOSURE, to effectively and efficiently detect domain names that are involved in malicious activity. We use 15 features (9 of which are novel and have not been proposed before) that allow us to characterize different properties of DNS names and the ways that they are used (i.e., queried).

Note that researchers have used DNS before as a way to analyze, measure and estimate the size of existing botnets in the past (e.g., [48, 56, 93]). Some solutions have then attempted to use DNS traffic to detect malicious domains of a certain type (e.g., [78,100]). However, all these approaches have only focused on specific classes of malware (e.g., only malicious Fast-Flux services). Our approach, in comparison, is much more generic and is not limited to certain classes of attacks (e.g., only botnets).

In our approach, based on features that we have identified and a training set that contains known benign and malicious domains, we train a classifier for DNS names. Being able to passively monitor real-time DNS traffic allows us to identify malware domains that have not yet been revealed by pre-compiled blacklists. Furthermore, in contrast to active DNS monitoring techniques (e.g., [100]) that probe for domains that are suspected to be malicious, our analysis is stealthy, and we do not need to trigger specific malicious activity in order to acquire information about the domain. The stealthy analysis that we are able to perform has the advantage that our adversaries, the cyber-criminals, have no means to block or hinder the analysis that we perform (in contrast to approaches such as in [100]).

To date, only one system has been proposed that aims to detect malicious domains generically using passive DNS analysis. In a concurrent and independent work that was very recently presented by Antonakakis et al. [14], the authors present Notos. Notos dynamically assigns reputation scores to domain names whose maliciousness has not been discovered yet. In comparison, our approach is not dependent on large amounts of historical maliciousness data (e.g., IP addresses of previously infected servers), requires less training time, and unlike Notos, is also able to detect malicious domains that are mapped to a new address space each time and never used for other malicious purposes again.

1.3 Botnet Command and Control Server Detection Through Netflow Analysis

Although EXPOSURE performs well in detecting botnets that utilize DNS names for contacting the C&C server, one current problem of the system is that it is evadable if the botnet's DNS usage is designed to be similar to benign servers' (e.g., normal TTL values, a mapping to a small number of IPs, etc.). A second problem with DNS-based detection is that a request from the client to a malicious DNS domain does not necessarily indicate that the client has been infected. In fact, such a request might have been simply caused by a failed infection attempt. Furthermore, there is also a chance that the queried domain is not really malicious (e.g., if the attackers are using a benign, compromised domain to host a C&C server). A third problem is that a DNS query for a malicious domain might be cached and hence, might not appear in the monitored DNS traffic. As a result, connections by the compromised hosts to the C&C server may be missed.

One other problem all previous network-based botnet detection techniques share is that they do not scale beyond a single administrative domain while retaining useful detection accuracy, even though, they are effective under certain circumstances. This limitation restricts the application of automated botnet detection systems to those entities that are informed or motivated enough to deploy them. Thus, we have the current state of botnet mitigation, where small pockets of the Internet are fairly well protected against infection while the majority of endpoints remain vulnerable.

This situation is not ideal. Botnets are an Internet-wide problem that spans individual administrative domains and, therefore, a problem that requires an Internet-scale solution. In particular, botnets can continue to wreak havoc upon the Internet despite the deployment of localized detection systems by focusing on propagation through less well-protected populations.

Two of the primary factors preventing the development of effective large-scale, wide-area botnet detection systems are seemingly contradictory. On the one hand, technical and administrative restrictions result in a general unavailability of raw network data that would facilitate botnet detection on a large scale. On the other hand, were this data available, real-time processing at that scale would be a formidable challenge. While the ideal data source for large-scale botnet detection does not currently exist, there is, however, an alternative data source that is widely available today : NetFlow data [35].

NetFlow data is often captured by large ISPs using a distributed set of collectors for auditing and performance monitoring across backbone networks. While it is otherwise extremely attractive, NetFlow data imposes several challenges for performing accurate botnet detection. First, and perhaps most critically, NetFlow records do not include packet payloads ; rather, flow records are limited to aggregate metadata concerning a network flow such as the flow duration and number of bytes transferred. Second, NetFlow records are half-duplex ; that is, they only record one direction of a network connection. Third, NetFlow data is often collected by sampling the monitored network, often at rates of several orders of magnitude or more, removed from real traffic.

Each of these characteristics of NetFlow data complicates the development of an effective botnet detector over this domain. The detector must be able to distinguish between benign and malicious network traffic *without* access to network payloads, which is the component of network data that carries direct evidence of malicious behavior. The detector must also be able to recognize weak signals indicating the presence of a botnet due to the combined effects of half-duplex capture and aggressive sampling.

As a novel follow-up work to our previous botnet detection techniques, in the third part of the thesis we present Disclosure, a large-scale, wide-area botnet detection system that incorporates a combination of novel techniques to overcome the challenges imposed by the use of NetFlow data. In particular, we identify several groups of features that allow Disclosure to reliably distinguish C&C channels from benign traffic using NetFlow records : (i) flow sizes, (ii) client access patterns, and (iii) temporal behavior. We demonstrate that these features are not only effective in detecting current C&C channels, but that these features are relatively robust against expected countermeasures future botnets might deploy against our system. Furthermore, these features are oblivious to the specific structure of known botnet C&C protocols.

While the aforementioned features are sufficient to capture core characteristics of generic C&C traffic, they also generate false positives in isolation. To reduce Disclosure's false positive rate, we incorporate a number of external reputation scores into our system's detection procedure. These additional signals function as a filter that reduces DISCLOSURE's false positive rate to a level where the system can feasibly be deployed on large-scale networks.

1.4 Contributions

In summary, this thesis makes the following contributions :

- We present three network-based botnet detection techniques that perform their analysis on different types of data : full network traffic, DNS traffic and NetFlow traffic.
- We present a botnet detection system that performs network level packet inspection. This system is a fully automated mechanism that generates bot detection models by observing the actual behavior of bot instances in a controlled environment, without making assumptions about the C&C mechanisms. This work has been published in ESORICS 2009.
- With the experiment we performed with the first detection method, we demonstrate the feasibility of our approach by generating detection models for various bot families (including those controlled via IRC and HTTP, as well as P2P). These models are effective in detecting bots with few false positives.
- We present another novel analysis technique for the detection of malicious domains that is based on passive DNS request analysis. Our technique does not rely on prior knowledge about the kind of service the malicious domain provides (e.g., phishing, Fast-Flux services, spamming, botnets that use a domain generation algorithm, etc.). This is significantly different from existing techniques that only target Fast-Flux domains used in botnet operations. Furthermore, our approach requires less training time, and less training data than Notos [14], and does not have some of its limitations. This work has been published in NDSS 2011.
- We describe the implementation of our real-time detection system which we call EXPOSURE. Our experimental results show that the technique we propose is scalable, and is able to accurately distinguish between malicious and benign domains with a low false positive rate.
- We present DISCLOSURE, a large-scale, wide-area botnet detection system that reliably detects botnet C&C channels in readily-available NetFlow data using a novel set of robust statistical features. In partic-

ular, DISCLOSURE does not assume *a priori* knowledge of particular C&C protocols.

- We incorporate several external reputation systems into DISCLOSURE's detection procedure to further refine the accuracy of the system.
- We evaluate DISCLOSURE over two real-world networks, and demonstrate its ability to detect both known and unknown botnet C&C servers at scales not previously achieved.

1.5 Outline

This dissertation is organized as follows :

Chapter 2 : The malware evolution : Botnets
In this chapter, we explain the botnet infrastructures, command and control topologies and the historical botnet evolution.

Chapter 3 : State of The Art In this chapter, we explain the most recent work on botnet related malware analysis topics.

Chapter 4 : Botnet Detection Through Network Level Packet Inspection In this chapter, we present our first botnet detection technique. This technique performs network-level packet inspection to generate detection models for bot infected machines.

Chapter 5 : Botnet Detection Through Passive DNS Analysis
In this chapter we explain EXPOSURE, a real-time detection system that aims to identify domains that are abused for nefarious purposes such as being used for a botnet C&C server, as a dropzone to collect the sensitive information sent by bots.

Chapter 6 : Detecting Botnet Command and Control Servers Through NetFlow Analysis In this chapter we explain DISCLOSURE, a system that detects botnet C&C servers performing NetFlow analysis.

Finally, in Chapter 7, concluding remarks are given.

Chapitre 2

The Malware Evolution : Botnets

A botnet is a network that consists of a large number of compromised machines that have the ability to be controlled by malicious entities. These compromised machines are typically termed as bots. They are also known as zombies or drones. The botnet controller (i.e. botmaster, bot herder) sets a command and control channel (C&C channel) to command her bots to perform malevolent activities such as sending spam, launching denial of service attacks, stealing sensitive information from the user of the compromised machine. Unlike other types of attacks, botnets which may consist of thousands of compromised hosts can assemble a tremendous volume of aggregate computing power and can perform a variety of attacks against a wide range of targets.

In many respects, the botnets found in the wild today are a hybrid of previous malware. However, the C&C mechanism they have distinguishes them from other type of malware. The command and control protocol not only allows the attackers to remotely control their bots for nefarious purposes but also gives them a great flexibility to change their malicious infrastructure completely against emerging botnet detection mechanisms.

Botnets typically infect victim computers with many different techniques. Most typical infection schemes include exploiting known vulnerabilities, luring the Internet users to click links by applying social engineering techniques and drive-by-downloads. Botnets inherits the propagation mechanisms from worms which are known to be their ancestors.

2.1 Botnet Characteristics

Today, the botnet characteristics are still not well understood. Botnets are still continuing their evolution, therefore accepted as an emerging threat. Botnets exhibit all characteristics of other classes of malware such as worms, trojan horses, rootkits etc. However, their defining characteristic is the communication channel established between the botmasters and the bots. In the following sections, while we will mostly focus on the communication topologies botnets have, we will also briefly explain other characteristics such as bot propagation, exploitation and attack mechanisms.

2.1.1 Botnet Command and Control Infrastructures

The botnet command and control infrastructure gives the bots the ability to communicate with the botmaster(s). In the wild, there are many different type of botnets that vary in size, in malicious activities they perform and in network topologies they construct. Consequently, they use various C&C topologies.

Most of the botnet detection techniques leverage the existence of the command and control action in the network. While some try to identify the command and control channel to find the location of the C&C server, others tackle the problem of finding the infected machines (i.e. bots). That is, the component of botnets that is attacked by botnet defense mechanisms is the command and control protocol. The response of botnet creators for evasion is to enhance their command and control topology. Therefore, the botnet evolution is realized through the enhancements made on the command and control topology.

The command and control topologies that are widely employed can be categorized in two main groups : centralized and decentralized command and control. Star, multi-server and hierarchical C&C topologies fall in to the centralized C&C topologies. On the other hand, the random C&C topology is decentralized.

– **Star C&C Topology.** Star topology has a centralized C&C server in botmaster's control as seen in Figure 2.1. All bot agents connect to this server to receive the commands. The typical behavior of newly bot infected machine is to establish the connection with the C&C server to join the botnet. This preconfigured behavior is also known

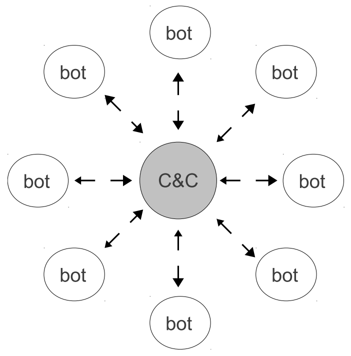


Figure *2.1: Star C&C Topology*

as "phoning home". Employing star topology allows the botnet an efficient command and control transfer. However, if the server fails, the botnet fails. Therefore, it is very vulnerable for the shutdowns. IRC bots and HTTPs usually employ the star topology.

– **Multi-Server C&C Topology.** Multi-Server C&C Topology is an extended version of star C&C topology. It has multiple C&C servers that are responsible for maintaining subsets of the botnet as seen in Figure 2.2. The C&C servers in the topology communicate among themselves to integrate the sub-botnets. This topology is more robust than the star topology since it is more tolerant to the failures. If a number of the severs fail, the remaining servers maintain the integrity. Typically, the botmasters distribute the C&C servers in different countries such that the bots in those locations can communicate with the server in an efficient manner.

– **Hierarchical C&C Topology.** In this topology, there is a hierarchy among the bots. At the infection phase, the bot either transforms to a proxy or a bot that contributes to the malicious activities commanded by the botmaster. The proxy bots are responsible for forwarding the commands they receive to the botnet. The components the bots are in contact with in the botnet are the proxy bots and not the botmaster. The real C&C servers are hidden behind the proxies as it is in Figure 2.3. Therefore, the bots are not aware of the rest of the botnet and where the C&C server is located.

  Botnets that employ hierarchical C&C topology are very difficult to take down and reverse-engineer to make estimations about the size and the structure of it. Because, even if one of the bots is captured by researchers, it is impossible to get more information than the IP

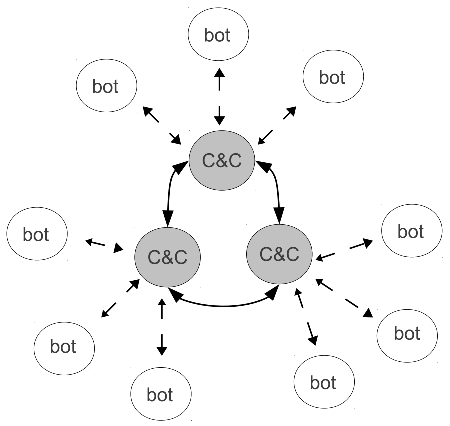


FIGURE *2.2: Multi-Server C&C Topology*

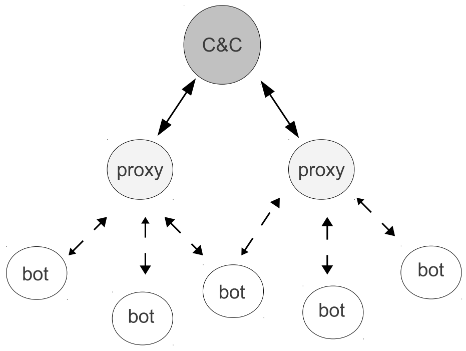


FIGURE *2.3: Hierarchical C&C Topology*

address or the domain name of the responsible proxy bot.
Hierarchical C&C topologies also allows the botmaster to split the botnet to sub-botnets such that she can rent or sell services to other botmasters. Clearly, this type of topology brings several advantages to the botnet owners. However, there can be an acceptable latency during transmitting the command to the bots due to the fact that the commands traverse through multiple points. This delay makes some malicious activities difficult to be realized.

– **Random C&C Topology.** The random C&C topology is not a centralized topology as the topologies we listed above, but decentralized as seen in Figure 2.4. The commands are not distributed from a central point. All bots play a role on sending the command to the botnet. When a bot receives a command, it immediately triggers the command

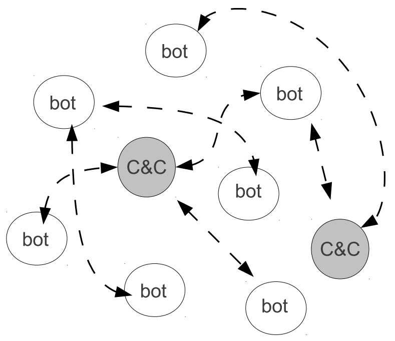


FIGURE *2.4: Random C&C Topology*

propagation module that transmits the command to its neighbors. One good example for botnets with random C&C topologies are peer-to-peer botnets.

Botnets with random C&C topology are more flexible for shutdowns because they do not have a centralized C&C server and multiple paths to conduct the commands. However, once one of the bot agent is compromised by researchers or malware analyzers, by reverse-engineering, it is relatively easier to identify other members of the botnet. The reasoning behind this is that if the malware analyzers are able to reverse-engineer the algorithm, they can join the botnet and start communicating with the other bot infected machines.

2.1.2 Bot Propagation Mechanisms

One crucial phase in lifecycle of botnets is the infection phase. The malware infection might be realized in different ways : exploiting vulnerabilities on the host, luring the user into clicking a malicious URL, and employing social engineering techniques. Among these infection techniques, infection through vulnerability exploitation does not require a user interaction.

Typically botnets that propagate through exploiting known or zero-day vulnerabilities include a scanning component that is responsible for finding new vulnerable machines to be infected. There exists several scanning mechanisms that can be categorized into two groups : horizontal and vertical scanning mechanisms. Horizontal scanning mechanism scans a single point in a specified address space. In contrast, vertical scanning mechanisms scan a port range on a single IP address. In order to find the IP address or the range to be scanned, the flowing methods are widely used :

– **Random Scanning :** The target to be scanned is determined by a random number generator. Thus, the effectiveness of the scanning strictly depends on the random number generator. Since it is quite difficult to develop a random number generator that can find vulnerable hosts or valid IP addresses, random scanning is not as effective as other scanning methods.

- **Permutation Scanning :** One reason why random scanning is not effective enough is because of the overlap it produces. Permutation Scanning was designed to deal with the problem of overlaps. It applies simple cryptography such that each malware sample can generate different sets of target IP addresses. The list of IP addresses are computed by a pseudo - random permutation function that takes a private key as parameter. In this way, permutation scanning solves the overlap problem by using cryptography.
- **Hit-List Scanning :** Hit-list scanning is the most efficient scanning technique. This is because the hit-list comes hard-coded in the malware. Since the binary includes the list, the size of the binary can be unacceptably large. During the spreading process, the IP addresses already scanned are removed from the list in the binary. Therefore, the size of the binary decreases in time. Obviously, since each malware sample receives different snippets from the list, hit-list scanning does not have the overlapping problem.
- **Combining the Techniques :** Some of the more sophisticated worms seen in the wild, such as $Warhol$ worm, employ a combination of scanning techniques. For example, Warhol by combining permutation scanning and hit-list scanning was able attack all vulnerable machines in the world in less than fifteen minutes.

Although current botnets employ more sophisticated techniques to find and infect their victims, historically, very famous botnets such as Agobot, SDBot, SpyBot and GTBot had fairly simple propagation mechanisms that apply vertical and horizontal scanning. In order to detect such botnets, it is possible to develop statistical finger printing methods to identify bot scans.

2.1.3 Malicious Activites

The primary goals of botnets include information dispersion, information harvesting and information processing. Information dispersion attacks includes sending spam, launching distributed denial of service attacks and click fraud. On the other hand, information harvesting involves attacks that steal sensitive information from the infected hosts to obtain identity data, financial data, private data, e-mail address books or any other type of data may exist on the host. Main intention of information processing is to use the distributed structure of botnets to crack passwords, hashes etc.

Although some botmasters construct their botnets for fun or fame, the majority of current botnets do for financial gain. For example, an organization that needs to advertise its products might wish to pay a botmaster to make its advertisements sent through spam. As another example ; the stolen secret information might be directly used by the botmaster or sold to third parties.

Distributed Denial of Service Attacks

Botnets are widely used to perform distributed denial of service attacks (DDoS). A DDoS attack is an attack that targets either a computer or a network to make a resource unavailable to its users. Typically, the loss of the service or network connectivity is realized by consuming the bandwidth of the network or overloading the network stack of the computer. A DDoS attack can be performed in a number of different ways. Some of these techniques are listed as :

– Consuming the computational resources, e.g. bandwidth, disk space or processor time.
– Corrupting the configuration information such as routing table configuration.
– Disrupting the state information, such as unsolicited resetting of TCP sessions.
– Obstructing the communication media in order to prevent the users from communicating each other.

Today, it is very easy to mount DDoS attacks with the help of off-the-shelf tools [44] . There are different kinds of attacks that target the TCP, the UDP or protocols at higher level in the network stack :

1. **TCP SYN flooding :** TCP SYN flooding attack is performed by sending several connection requests to target computer in order to stress the processing ability. The half open connections on the target machine exhaust the data structures in the kernel. Thus, the computer cannot accept new connections.
2. **UDP flooding :** The attacker aims to consume the network bandwidth and computational resources by sending a large number of UDP packets to several ports.
3. **DDoS attacks targeting high-level protocols :** DDoS attacks are the most dangerous form of DoS attacks. They are not only restricted

to the web services. By creating more specific attacks that target high-level protocols, more effective results can be obtained. The *web spidering attack*, which starts from a given web site and then recursively requests all links on that site, is a good example for DDoS attacks that target high-level protocols.

During last ten years, Internet users faced several serious DDoS attacks. In February 2000, an attacker launched DDoS attacks on several e-commerce companies and web sites. The attacks deactivated the service of the servers for several hours. Later, the threat of the DDoS attacks turned into real cybercrime. For example, a botnet targeted a betting company during the European soccer championship in 2004 and demanded money in exchange of letting the system operate again.

Spam

E-mail spamming, a.k.a. bulk e-mail or junk e-mail, is an attack that sends massive volumes of nearly identical messages to recipients by e-mail. Generally, such messages have commercial content. An e-mail is spam only if it is unsolicited and sent in bulk. E-mail addresses used by the spammers are collected by chat rooms, newsgroups, websites and the malware that harvest e-mail addresses from the users' address books.

It has been reported that 80% of the spam e-mails are sent by botnets. Typically, bots start a SOCKS v4/v5 proxy on the compromised host in order to use it for sending spam e-mails. Once they receive the spaming activation command by the botmaster, thousands of members of the botnet initiate a large-scale spam activity on the Internet.

In order to identify valid mail addresses, some botnets concentrate only on harvesting valid e-mail addressed. Such botnets sell the lists they gathered to spammer botnets. Besides collecting the infected users' e-mail addresses, botnets crawl web to gather e-mail addresses from web sites, newsgroups, special-interest group (SIG) postings, and chat-room conversations.

Click Fraudulence

In addition to the malicious activities that directly are involved in attacks, botnets can perfom click fraudulence. Click fraudulence is performed by automatically and periodically querying particular websites to increase the number of clicks to increase the search scores of the website or manipulate the votes.

Identity Theft

To steal that sensitive information from infected hosts, botnets install keyloggers or sniff the traffic. Moreover, they install simple programs that collects usernames and passwords from the host, and by sniffing the network traffic passwords of other users in the network.

Phishing Mails

Phishing is one type of identity theft attack which aims to compromise sensitive information (e.g., passwords, credit card numbers) by masquerading as a trustworthy entity in an electronic communication. Phishing attacks use sophisticated social engineering techniques to persuade users to give their secret information. There are different types of phishing attacks :

- **Spoofing Mails and Web Sites :** The earliest phishing attacks were e-mail based. The attackers were trying to persuade the victim users to send their passwords and account information by sending spoofed e-mails. Although there are still many users that can be deceived by phishing mails, most users are aware that sensitive information must not be sent by e-mails. Thus, the attackers employed more sophisticated phishing techniques to deceive the victims. For example, some phishing attacks combine phising mails and websites by sending the URL of the phishing web page in mail that apper to come from a legitimate orginization. After the user clicks the link in the mail, the e-mail directs the user to a web site that looks identical to a familiar web site. Then, the user perform his normal actions, such as logging into the site or sending account information. Clearly, this reveals all the secret information to the attacker.
- **Exploit Based Phishing Attacks :** Exploit Based Phishing Attacks are more complicated in their nature, because, they exploit known vulnerabilities to install a backdoor. The backdoor collects the sensitive information and sends it back to the attackers.

2.2 Historical Evolution of Botnets

Today, botnets are known to be the most sophisticated malware. Ironically, the earliest bots were invented for benign purposes. These bots operated on the IRC [55] network, which was developed in the late 1980s. The IRC platform allows data dissemination among a large number of users with point-to-point or point to multi-points communication schemes. The bots created in the IRC network were used for entertaining users by offering them game and messaging services.

Unfortunately, not long after, the first malicious botnet, GTbot, emerged in the wild. The command and control infrastructure GTbot had was based on Microsoft IRC client, *mIRC.exe*, with slight modifications. Afterwards, in late 1999, SANS Institute researchers discovered remotely executable code on thousands of Windows machines. They were inspired by remote control nature of the code while they were naming the infected computers as *robots*, which later is shortened to *bot*.

Before 2004, most of the famous botnets such as SDbot, Agobot, Spybot had IRC-based command and control infrastructures. However, existing effective botnet detection methods against IRC-based botnets lead the botnets to evolve and build more sophisticated command and control topologies such as peer-to-peer (P2P) structures. Instead of using IRC as C&C protocol, they adapted different protocols (e.g., HTTP). Moreover, in order to be more robust and fault tolerant against shutdowns, they built Fast Flux Service Networks.

The most critical requirement for a botnet is to have a reliable C&C infrastructure to avoid it's bots transformed to zombies when connection with the C&C server cannot be established. The first botnets seen in the wild had centralized C&C servers whose IP addresses were hard coded in the bot binaries. Such botnets were very vulnerable for the shutdowns. Because, to take the complete botnet down, it was enough to blacklist the IP address of the C&C server or shut it down.

Botnet designers developed a number of techniques to provide a malicious infrastructure whose members (i.e., bot agents) remain always connected to the botnet. These techniques not only resulted the botnet evolution but also made the botnets more resilient and robust against shutdowns.

The most effective technique for increasing the lookup resilience is called fluxing. The fluxing can be implemented in two ways : IP fluxing and domain fluxing. IP fluxing is realized by assigning a large amount of IP addresses to a single domain name. The botnets that employ IP fluxing abuse the Round Robin feature of Domain Name System (DNS). Typically the botnet operators set the Time-to-Live (TTL) value of the domain to a lower value (e.g. less than 300 seconds) and each time the domain receives a query a different list of IP addresses is returned.

There exists two types of IP fluxing methods : single-flux and double-flux. Single-flux employs the simple IP fluxing method. That is, a domain name is associated with hundreds or even thousands of IP addresses and each time a different set of them is returned with in the DNS record. Double-flux, which is a more sophisticated version of single-flux, fluxes both the IP addresses of the domain name and the IP addresses of the DNS server.

Domain flux, on the other hand, associates a large number of domain names with a single or a few IP addresses or the C&C servers. Domain fluxing is realized either by domain wildcarding or domain generation algorithms. The domain wildcarding abuses the native DNS option to wildcard (i.e., *) a higher domain level and associates all sub domains to the same IP address. This features is employed mostly by spambots and botnets that apply phishing techniques in their attacks to hide the sub domains that include randomly generated strings. During the recent years, botnets started to apply domain fluxing techniques in which the bot agents generate the domain name to be contacted at a specific time by a domain generation algorithm (DGA). The basic idea is that the botnet operators register a large number of domain names and associate each of them with the C&C server at a specific time. The domain generation module that comes within the bot binary generates the domain of the C&C server taking the current time as parameter. The domains that are generated by a DGA typically have very short life such as one day. Therefore, it is very hard to investigate these domains to track the location of the C&C servers. Conficker botnet [80] that was one of the ten most wanted botnets in 2009 and 2010 deploys DGA modules to its bot agents.

Chapitre 3

State of the Art

Malware, and botnets in particular, pose a significant threat to the security of the Internet. As a result, there has been a strong interest in the research community to develop adequate defense solutions. Because this thesis proposes three botnet detection techniques that perform analysis on network packets, DNS data and NetFlow data, it naturally touches on a number of related research areas.

3.1 General malware detection

Since bots are a certain type of malware, previous work in identifying malicious code is directly relevant. The most common approach to fight malicious software are commercial anti-virus scanners. Anti-virus scanners mostly rely on syntactic signatures that match parts of known malware samples. As a result, these systems cannot identify previously unknown malware programs or simple variations of existing ones [32].

To address the limitations of signature-based malware detection, researchers have proposed behavior-based techniques. These techniques attempt to characterize a program's behavior in a way that is independent of its binary representation. Christodorescu et al. use static program analysis techniques to identify semantically equivalent operations in malware variants [33]. Because malware authors can use code obfuscation techniques or self-modifying code, static malware analysis is difficult [72]. To address this shortcoming, researchers have proposed dynamic approaches [59, 111] that monitor the execution of a program to identify malicious behaviors.

While the aforementioned techniques are powerful in detecting even previously unknown malware instances, they require extensive analysis of each suspicious program. This not only incurs a considerable performance overhead, it also requires that the analysis system is installed on every machine that should be protected. With the systems presented in this thesis, we provide network-based solutions that are more efficient compared to host-based malware detection systems.

3.2 Network intrusion detection

The purpose of network intrusion detection systems (IDS) is to monitor the network for the occurrence of attacks. Clearly, this is very similar to the purpose of our first detection technique that analyze network traffic for the presence of signs that indicate bot-infections. In fact, we directly encode our detection models in the signature language of Bro [77], a well-known, network-based IDS.

Of course, both the ideas of content-based analysis and modeling network-level properties to detect anomalies are not novel. Content-based analysis has been used by signature-based IDSs (such as Snort [88] or Bro) for years. Also, network-level properties (such as the number of flows or the number of bytes that were transferred) have been used extensively to model normal network traffic and to detect deviations that indicate attacks [69]. Our work complements existing network-based IDSs by automatically generating the inputs needed by these systems to detect machines that are infected by bots.

3.3 Signature generation

As part of our detection model generation in our first botnet detection system, we extract token signatures from network traffic. Research on such automated signature generation started with the work on Early Bird [89] and Autograph [58], and has later been extended with Polygraph [75] and Hamsa [63]. Of course, extracting command tokens is only a small part of the entire model generation process. In fact, we first have to record bot activity, identify likely bot responses, extract the corresponding traffic snippet, and cluster them based on behavioral similarities. Only then can we extract common tokens, using an improved version of previous algorithms.

3.4 Botnet analysis and defense

In addition to general research on malware detection, there is work that specifically focuses on the analysis [36, 48, 54, 83] and detection [26, 49, 51–53, 56, 86, 95] of botnets.

A number of botnet detection systems perform horizontal correlation. That is, these systems attempt to find similarities between the network-level behavior of hosts. The assumption is that similar traffic patterns indicate that the corresponding hosts are members of the same botnet, receiving the same commands and reacting in lockstep. While initial detection proposals [53, 56] relied on some protocol-specific knowledge about the command and control channel, subsequent techniques [51, 86] remove this shortcoming. The main limitation of systems that perform horizontal correlation is that they need to observe multiple bots of the same botnet to spot behavioral similarities [1]. This is significant because botnets decrease in size [36], it becomes more difficult to protect small networks, and a botmaster can deliberately place infected machines within the same network range into different botnets.

A second line of research explored vertical correlation, a concept that describes techniques to detect individual bot-infected machines. One system, called Rishi [49], attempts to detect bots based on the structure of nicknames in IRC traffic. Other techniques [26, 95] aim to identify suspicious IRC connections based on traffic properties. In all cases, the detection approaches focus specifically on botnets that use IRC for their command and control. The most advanced system is BotHunter [52], which correlates the output of three IDS sensors – Snort [88], a payload anomaly detector, and a scan detection engine. A closer analysis of the results (which are publicly available [39]) reveals that the detection capability of BotHunter strongly relies on the human-created Snort rules.

The first botnet detection technique presented in this thesis performs vertical correlation as well as BotHunter does. Our technique, on the contrary to BotHunter, generates detection models completely automated. Moreover, the stages that are used by BotHunter to characterize the life cycle of a bot focus on scanning and remote exploiting. Our system, on the other hand, does not rely on a specific bot propagation strategy and does not require previous knowledge about command and control channels.

Independently and concurrently to our work, a paper [54] has presented the idea of running bots in a controlled environment (called Botlab). The

1. With the exception of a narrow, special case presented in [53].

proposed system is similar to ours in that bots are executed and monitored. The difference is that Botlab is exclusively focused on spam botnets and uses the monitored activity (in addition to other inputs) to produce information about spam mails (such as malicious URLs in the mail body). However, the approach does not provide any information about bot commands or responses, and it is not designed to detect bot infected machines. Thus, the goals are very different from our system.

3.5 Using DNS Analysis Techniques for Detecting Botnets

The Domain Name System (DNS) has been increasingly being used by attackers to maintain and manage their malicious infrastructures. As a result, recent research on botnet detection has proposed number of approaches that leverage the distinguishing features between malicious and benign DNS usage. The second botnet detection technique (EXPOSURE) presented in this thesis uses passive DNS analysis to identify domains that are involved in malicious activities such as being the command and control server of a botnet.

The first study [105] in this direction proposed to collect real-world DNS data for analyzing malicious behavior. The results of the passive DNS analysis showed that malicious domains that are used in Fast-Flux networks exhibit behavior that is different than benign domains. Similarly, Zdrnja et al. [112] performed passive monitoring to identify DNS anomalies. In their paper, although they discuss the possibility of distinguishing abnormal DNS behavior from benign DNS behavior, the authors do not define DNS features that can be used to do so.

In general, botnet detection through DNS analysis follows two lines of research : The first line of research tries to detect domains that are involved in malicious activities. The goal is to identify infected hosts by monitoring the DNS traffic. The second line of research focuses on the behaviors of groups of machines in order to determine if they are infected (e.g., a collection of computers always contact the same domain repeatedly).

3.5.1 Identifying Malicious Domains

To detect malicious domains, previous approaches make use of passive DNS analysis, active DNS probing, and WHOIS [2] information. For example, recent work by Perdisci et al. [78] performs passive DNS analysis on recursive DNS traffic collected from number a number of ISP networks with the aim of detecting malicious Fast-Flux services. Contrary to the previous work [61, 73, 76, 100], Perdisci's work does not rely on analyzing blacklisted domains, and domains that are extracted from spam mails. EXPOSURE significantly distinguishes itself from theirs as we are able to detect all different kinds of malicious domains such as phishing sites, spamming domains, dropzones, and botnet command and control servers. We do not only focus on detecting Fast-Flux service networks.

A second branch of study that aims to detect malicious domains [68, 100] leverages active DNS probing methods. That is, the domains that are advertised to be malicious by various sources (e.g. spam mails) are repeatedly queried to detect the abnormal behavior. The main drawback of active DNS analysis is the possibility of being detected by the miscreants who manage the domains under analysis. Passive DNS analysis, in comparison, is more stealthy because of its non-intrusiveness characteristics.

Based on URL features they extract from spam mails, Ma et. al. [68] study a number of statistical methods for machine learning for classifying websites. In particular, they analyze spam URLs according to their lexical construction, and the information contained in the host name part of the URL. To obtain the information from the host name, they perform active probing to determine the number of IP addresses associated with the domain. Once they obtain the IP address list, they analyze the location of the IP address and to which ASN it belongs to. The main limitation of this system is that it performs the analysis only based on the domains that are included in spam mails. Hence, the system cannot see other classes of malicious domains such as command and control servers.

Another type of study on detecting malicious domains leverages properties inherent to domain registrations and their appearance in DNS zone files [46]. That is, they associate the registration information and DNS zone properties of domains with the properties of known blacklisted domains for proactive domain blacklisting. This method completely relies on historical information. Therefore, it is not able to detect domains that do not have any registration information and DNS zone commonalities with known blacklisted domains. On the other hand, our work, which does not require any historical information, is able to detect such domains.

3.5.2 Identifying Infected Machines by Monitoring Their DNS Activities

In [31], the authors propose an anomaly-based botnet detection mechanisms by monitoring group activities in the DNS traffic of a local network. The authors claim that there exist distinguishing features to differentiate DNS traffic generated by botnets and benign clients. Similarly, [102] also attempts to identify botnet DNS access behavior in a local network. The authors use a bayesian algorithm. In comparison to these existing works, we aim to identify malicious domains from DNS traffic in general, and do not only focus on botnets.

3.5.3 Generic Identification of Malicious Domains Using Passive DNS Monitoring

To date, there are two systems proposed that aim to detect malicious domains using passive DNS analysis as well as EXPOSURE. In a concurrent and independent work presented by Antonakakis et al. [14], the authors present Notos. Notos dynamically assigns reputation scores to domain names whose maliciousness has not been discovered yet.

We have compared EXPOSURE with Notos in the evaluation section of Chapter 5. EXPOSURE eliminates several shortcomings of Notos. It does not require a wide overview of malicious activities on the Internet, a much shorter training time, and is able to classify domains that Notos would miss-classify.

The second work was proposed after EXPOSURE and Notos by Antonakakis et. al. [15]. The malware domains detection system, which is named as Kopis, employs DNS data collected from upper DNS hierarchy. Therefore, they were able to analyze global DNS query resolution patterns. Kopis leverages the fact that in their data the IP addresses of the clients who issued the DNS queries are visible. Although Kopis performs well to detect emerging new botnets, it cannot be deployed as a real-time botnet detection system on a local network. On the other hand, EXPOSURE can be also deployed in an independent network to monitor clients DNS activity.

3.6 Anomaly Detection Through NetFlow Analysis

To date, there has been a considerable amount of research on anomaly detection using NetFlow analysis. While some of the works proposed anomaly detection methods to detect specific kinds of malware such as worms [103], others tried to propose more general approaches to distinguish malicious traffic from benign traffic [29, 43, 90].

Wagner et al. [103] present an entropy-based approach to identify fast worms in real-time network traffic by calculating the entropies of traffic parameters such as IP addresses. They detect massive network activities by observing the changes in the entropy of the traffic. Dewaele et al. [43] extract sub-traces from randomly chosen traffic traces, model them using Gamma laws, and identify the anomalous traces by tuning the deviations in the parameters of the models. When the anomalous sub-traces are identified, the authors detect the anomalous source and destination IP addresses by analyzing flows that match the intersection of addresses that hash into anomalous sub-traces. Brauckhoff et al. [29] present a histogram-based anomaly detector that identifies anomalous flows by combining various information extracted from multiple histogram-based anomaly detectors.

Another NetFlow-based anomaly detection method for distinguishing malicious and benign network traffic was proposed by Sperotto et al. [90]. The authors analyzed the time series constructed from both flow and packet sizes, and tested them to find whether they were sufficient for detecting general intrusions. Their analysis results show that it sufficient to employ one type of time series for some attacks, but that multiple time series need to be combined for more accurate results.

Rehak et al. [85] propose a system that deploys trust modeling to decrease the high error rates network based anomaly detection techniques engender. The system deploys multiple agents each of which applies one of the existing anomaly based detection methods. The anomaly values determined by the agents are averaged to produce more accurate results.

Another line of research on NetFlow-based anomaly detection focuses on the analysis of the impact of sampling methods used to decrease the input data rate to a manageable volume. Mai et al. [70] analyze a set of sampling techniques applied for collecting, recording, or forwarding NetFlow traffic to identify the best sampling rate for the most accurate anomaly detection. The authors experiment with two classes of anomalies (i.e., volume anomaly and port-scans). The results of their evaluation show that all types of sampling techniques introduce a significant bias on anomaly detection. Another work [30] studied the impact of packet sampling on anomaly detection metrics such as the number of bytes, packets, and flows. The authors claim that packet sampling does not affect anomaly detection metrics such as the number of bytes or the number of packets, but that it significantly changes the number of flows. They also evaluate the impact of packet sampling and they conclude that entropy-based anomaly detection systems are more resilient to packet sampling because the sampling still preserves the distributional structure. Note that with DISCLOSURE, we performed two sets of experiments : a set with sampled NetFlows, and a set with unsampled NetFlows. We also show that even if the sampling degrades the anomaly detection in the specific case of botnet identification, it is still possible to distinguish malicious botnet traffic from benign ones if the NetFlow data set that is used is sufficiently large.

3.7 Botnet Detection with NetFlow Analysis

Only a few papers exist that propose to use NetFlow analysis to specifically detect botnets. For example, Livadas et al. [67] propose a system that identifies the command and control traffic of IRC-based botnets by using machine learning-based classification methods.

Francois et al. [47] present instead a NetFlow-based method that uses the PageRank algorithm to detect peer-to-peer botnets. In their experiments, the authors created synthetic bot traces that simulate the NetFlow behavior of three P2P botnet families.

Both works succeeded in the identification of a specific type of botnet traffic, IRC in the first case and peer-to-peer in the second. DISCLOSURE, on the other hand, can successfully detect C&C servers without any prior knowledge about the internals of the C&C protocol. Moreover, our experiments shows how DISCLOSURE can be used to perform real-time detection on large datasets.

Chapitre 4

Botnet Detection Through Network Level Packet Inspection

In this chapter, we present a system that aims to detect bots, independent of any prior information about the command and control channels or propagation vectors, and without requiring multiple infections for correlation. Our system relies on detection models that target the characteristic behavior of every bot – the fact that it receives commands from the botmaster to which it responds in a specific way. A key feature is that these detection models are generated automatically. To this end, our system observes the network traffic that is generated by actual bot instances in a controlled environment. In these traffic traces, we first identify points in time that likely correspond to response activity. Then, we extract the corresponding commands that trigger these activities. We have implemented the proposed approach and demonstrate that it can extract effective detection models for a variety of different bot families. These models are precise in describing the activity of bots and raise very few false positives.

4.1 System Overview

This section provides an overview of our approach to generate network-based detection models to identify bot-infected machines.

The input to our system is a collection of bot binaries. These binaries are collected in the wild, for example, via honeynet systems such as Nepenthes [16], or through Anubis [19], a malware collection and analysis platform. The output of our system is a number of models that can be used to detect instances of different bot families.

The basic idea of our system is to launch a bot in a controlled environment and let it connect to the Internet. Then, we attempt to identify the commands that this bot receives as well as its responses to these commands. Afterwards, these observations are translated into detection models that analyze network traffic for symptoms of bot-infected machines. The two main questions that arise are : (a) how are detection models specified, and (b), how can we generate these models based on observing bot activity ?

4.1.1 Detection Models

The goal of a detection model is to specify network traffic activity that is indicative of the presence of a bot-infected machine.

Stateful models. In our system, a detection model has two states. The first state of the model specifies signs in the network traffic that indicate that a particular bot command is sent. For example, such a sign could be the occurrence of the string `.advscan`, which is a frequently-used command to instruct an IRC bot to start scanning. Once such a command is identified, the detection model is switched into the second state. This second state specifies the signs that represent a particular bot response. Such a sign could be the fact that the number of new connections opened by a host is above a certain threshold, which indicates that a scan is in progress. When a model is in the second state and the system identifies activity that matches the specified behavior, a bot infection is reported. If no activity is found that matches the specification of the second state for some time period t, the model is switched back to the first state. Note that we maintain a different (logical) model instance for each host that is monitored. That is, when a command is found to be sent to host x, only the model for this host is switched to the second state. Therefore, there is no correlation between the activity of different hosts. For example, when a scan command is sent to host x, while immediately thereafter, host y initiates a scan, no alert is raised.

We make use of a stateful model that only labels a host as bot-infected if the system detects that a command is sent to the host **and** it witnesses a response within a certain period of time. This directly reflects the characteristic behavior of bots, which remotely receive commands from a botmaster and react accordingly. A stateful model has the advantage that we can use less restrictive specifications to capture both the command and the bot response, without risking an unacceptably high number of false positives. On the downside, our approach also represents a possible weakness that a botmaster can leverage to thwart detection. More precisely, a bot can delay the response to a command until after the detection model was switched back to the first state. This is a limitation that we share with previously-proposed systems [52, 53]. The reader is referred to Section 4.5 for a discussion of this limitation and possible solutions.

In general, there are two possible approaches to specify commands and responses in the different states of the detection model : content-based and network-based specifications. *Content-based specifications* are basically signatures. In its simplest form, we can provide a sequence of characters (i.e., a token) that has to be present in the network traffic. Of course, more sophisticated alternatives are possible, such as allowing regular expressions, or providing additional information about where a token is expected within the network stream. *Network-based specifications* focus on properties of the network activity that a host is engaged in. This is similar to anomaly-based intrusion detection systems. Simple models might only take into account the number of connections, while more sophisticated ones can also capture properties such as the number of characters that are transferred, the ports or protocols that are used, or entropy measurements of packets.

In our current system, we use content-based specifications to model commands and network-based specifications to model responses. This is a natural approach, where signatures capture commands and network models reflect the network activities due to responses (such as scanning, mass mailing, or binary downloads). However, observe that our general detection approach allows for alternative implementations. For example, one could use network-based properties to model commands. This might be advantageous when botnets use encrypted command and control channels. In that case, network-based properties could capture the frequency in which packets are exchanged, the size of packets, or the fact that encrypted payload is exchanged.

4.1.2 Model Generation

Given our notion of detection models, the question is how these models can be generated automatically. As mentioned previously, we do this based on the observation of bot activity. More precisely, for each bot binary, we first record a trace of its network activity over a certain period of time. Based on a trace, we have to identify those points where the bot receives a command and responds appropriately. This is not straightforward, since we do not want to assume any prior knowledge about the way in which the bots and the botmaster communicate. Also, it is possible and common that one trace contains a variety of commands with different responses. Thus, the problem of extracting detection models is quite different from previous signature generation algorithms [58, 63, 75] that simply extract common tokens from a set of network traces.

Finding responses. Our key insight for being able to identify previously unknown commands in a network trace is that we attack the problem from the opposite side. That is, instead of checking the traces for commands, we *first* look for the activity that indicates that a response has occurred. The reason for this approach is that a response launched by a bot is often more visible in the network trace than an incoming command. While a bot is in an idle state (i.e., it is not fulfilling requests of its botmaster), the network activity is typically limited to the traffic required to participate in the botnet (e.g., by exchanging IRC information or by polling web pages). However, when a command is issued, the bot has to act accordingly. This action almost always leads to additional network activity, for example, because the bot engages in scanning, downloads additional components, or sends mails. This activity stands out from the background noise and can be detected as an anomaly.

Once a bot response is identified, it is characterized by a *behavior profile*. More precisely, a behavior profile models various properties of the network traffic that is associated with a bot response. More details on recording bot traffic and locating responses are presented in Section 4.2.

Finding commands. By scanning the trace for network anomalies, we can identify those points in time at which a bot has demonstrated a response. As a result, the network traffic before this point must contain the command that has caused this response. Thus, before each point at which a significant change in traffic behavior is detected, we extract a *snippet*, a small section of the network trace.

Typically, different commands will lead to responses that are different. Therefore, in a next step, we cluster those traffic snippets that lead to similar responses, assuming that they contain the same command. Once clusters of related network snippets have been identified, we search them for sets of common (string) tokens. As our results demonstrate, these tokens frequently represent the bot commands and can be used for detection. Section 4.3 provides more details on the way in which traffic snippets are clustered and analyzed for common bot commands.

Putting it all together. Extracted tokens can be directly used to represent the bot command in the first state of the detection model. For the second state (i.e., to specify the response), we leverage the network behavior profiles that characterize bot response activity. Thus, in our current system, a bot detection model consists of a set of tokens that represent the bot command, followed by a network-level description of the expected response. These models can be readily deployed on the network and can identify an infected host once this host receives a known command and responds as expected.

Bot families. An important question that has been omitted so far is what happens when our system has to generate detection models for multiple bot binaries. The simplest approach is to analyze the network traces for each bot individually. In this case, only a single bot program is considered, and similar responses are very likely caused by the same command. However, this approach is not optimal for two reasons. First, when checking only a single trace, there might not be sufficient command-response pairs available to extract meaningful signatures. The second drawback is that due to botnet-specific artifacts (such as an IRC channel name) present in many of the snippets, the produced token signatures would likely match traffic from one specific botnet only. Thus, it is desirable to combine samples from different botnets into bot families, as long as they use the same C&C mechanism.

The partitioning of samples into bot families can be performed either manually, based on malware names assigned by anti-virus scanners, or based on behavioral similarities. For example, previous work has introduced host-based analysis systems that can find similar malware instances based on the system calls that these malware programs invoke [17, 20, 87]. Moreover, the partitioning step does not need to be perfect. Our system can tolerate the case in which the pool of bot network traces is polluted.

For the following discussion, we assume that the set of bot samples has already been divided into consistent groups. Of course, the system is neither provided with any information about the way in which commands are exchanged, nor how and when responses are launched.

4.2 Analyzing Bot Activity

As a first step to creating bot detection models, our system requires captures of the network traffic that the bot-infected machines create. To this end, we run each bot binary in a controlled environment with Internet access for a period of several days. The goal is to let the bot connect to its C&C mechanism and keep it running long enough to observe a representative collection of the different bot commands and the activities they trigger. The observed traffic should contain the most frequently used commands, since these are the most helpful detection targets. On the other hand, the absence of rarely used commands is acceptable, since detection models targeting these commands would also rarely trigger when deployed.

Note that we made a deliberate decision to observe the behavior of bots when they are connected to the actual botnet. This allows us to detect command and control traffic without any prior knowledge of the protocol and the commands that are used between the bot and the botmaster. Previous work [83] has proposed techniques to force bots to respond by exposing them to a barrage of likely IRC commands. While this might trigger a response quicker than in our case (and even provide the correct keyword), it only works when protocol and command tokens are known.

Further technical details regarding our solution for recording bot traffic traces are presented in the Appendix. In this section, we discuss how these traces are analyzed for the presence of response activity. Once the response activity is located, we can extract a snippet from the network traffic that precedes the start of the response and thus, likely contains the corresponding command. Moreover, we can collect behavior profiles, which describe the properties of the bot response behavior.

4.2.1 Locating Bot Responses

Once a network trace is collected, the next step is to locate the points within this trace where the bot executes responses to previously received commands. We do this by checking for sudden changes in the network traffic (e.g., a surge in the number of packets, or the fact that many different hosts are contacted). The assumption is that such changes indicate bot activity that is launched when a command is received. Of course, this implies that we can only detect bot responses (and hence, commands) that lead to a change in network behavior. However, most current bot responses, such as sending spam mails, executing denial of service attacks, uploading stolen information, or downloading additional components, fall into this category.

Of course, it is possible that there are changes in the traffic that are not caused by commands. For example, a scan might end when the list of victims is exhausted. Our system will also consider the end of the scan as a potential response, and mark the location appropriately. Fortunately, this is of little concern, because it is likely that the subsequent analysis will fail to find an appropriate command for this (inexistent) response. Sometimes, however, interesting detection models can be generated in such cases. For example, once a bot has finished scanning, it often sends a status notification to the botmaster, which can be recognized as an interesting content signature.

Locating bot responses in a network trace can be treated as a change point detection (CPD) problem. CPD algorithms operate on time series, that is, on chronologically ordered sequences of data values. Their goal is to find those points in time at which the data values change abruptly. Change point detection has been used previously to recognize spreading worms [110] and denial of service attacks [104]. However, we are not aware of any prior work that used it in the context of botnet detection.

To be able to apply a CPD algorithm, we first have to convert a traffic trace into a time series. To this end, the network traffic is partitioned into consecutive time intervals of equal length (our choice of a concrete interval length will be discussed later). Then, we compute a numeric description in form of a vector that represents the network traffic for each interval. For this, we extract a number of low-level features from the network traffic. Each feature captures a different aspect of the network traffic and translates into one element of the vector. Currently, we consider eight network traffic features :

1	Number of packets
2	Cumulative size of packets (in bytes)
3	Number of different IPs contacted
4	Number of different ports contacted
5	Number of non-ASCII bytes in payload
6	Number of UDP packets
7	Number of HTTP packets (destination port 80)
8	Number of SMTP packets (destination port 25)

TABLE *4.1: Network features to characterize bot behavior.*

Using the features shown in Table 4.1, we can characterize the bot's behavior during a given time interval. The characterization of bot activity is designed in a generic fashion, taking into account general features such as the number of packets, number of different machines contacted, or the number of (binary) bytes in network streams. In addition, we include two features that are derived from our domain knowledge of common bot responses : the numbers of SMTP and HTTP packets. The reason is that sending spam mails typically results in a surge of SMTP packets. The HTTP feature was initially considered as helpful to detect cases in which a bot downloads additional components via this channel. However, also currently unknown bot activity could be captured by our features, and it is certainly easy to add additional ones.

For every time interval, we calculate a vector that stores the absolute value for each feature. For example, when 50 packets are seen during a certain time interval, the corresponding element of the vector (number of packets) is set to 50. We call this vector a *traffic profile* of the bot for this time interval. To be able to compare behaviors obtained from different traces, this vector is normalized with regard to the maximum that was observed for the corresponding feature. This yields a value between 0 and 1 for all vector elements.

Change point detection. Once a network trace is converted into a sequence of traffic profiles, we apply a CPD algorithm to locate points that indicate interesting changes in the traffic. For this, we use CUSUM (cumulative sum), a well-known, robust algorithm that is known to deliver good results for many domains [18]. In principle, CUSUM is an online algorithm that detects changes as soon as they occur. Since we have the complete network trace (time series) available, we can leverage this fact and transform CUSUM into an off-line algorithm. This allows CUSUM to "look into the future" when a decision needs to be made, and thus, yields more precise results.

The algorithm to identify change points works as follows : First, we iterate over every time interval t, from the beginning to the end of the time series. For each interval t, we calculate the average traffic profile P_t^- for the previous $\epsilon = 5$ time intervals and the traffic profile P_t^+ for the subsequent ϵ intervals. Then, we compute the distance $d(t)$ between P_t^- and P_t^+. The distance between two traffic profiles is defined as the Euclidean distance between the corresponding vectors. More precisely :

$$P_t^- = \sum_{i=1}^{\epsilon} \frac{P_{t-i}}{\epsilon} \qquad P_t^+ = \sum_{i=1}^{\epsilon} \frac{P_{t+i}}{\epsilon} \qquad d(t) = \sqrt{\sum_{1}^{dim} \left| P_t^- - P_t^+ \right|^2} \tag{4.1}$$

The ordered sequence of values $d(t)$ forms the input to the CUSUM algorithm. Intuitively, a change point is a time interval t for which $d(t)$ is sufficiently large and a local maximum.

The CUSUM algorithm requires two parameters. One is an upper bound ($local\_max$) for the normal, expected deviation of the present (and future) traffic from the past. For each time interval t, CUSUM adds $d(t) - local\_max$ to a cumulative sum S. The second parameter determines the upper bound ($cusum\_max$) that S may reach before a change point is reported. To determine a suitable value for $local\_max$, we require that each individual traffic feature may deviate by at most $allowed\_avg\_dev = 0.04$. Based on this, we can calculate the corresponding value $local\_max = \sqrt{dim \times allowed\_avg\_dev^2}$. For $cusum\_max$, we use a value of 0.25. We empirically determined the values for $allowed\_avg\_dev$ and $cusum\_max$. However, note that these values are robust and yield good results for a large variety of traffic produced by hundreds of different malware instances that belong to different bot types (IRC, HTTP, and P2P bots).

It is possible that the cumulative sum S exceeds $cusum\_max$ for a number of consecutive time intervals. To locate the actual change point in this case, we take that interval for which $d(t)$ is maximal (since it is the time interval with the greatest discrepancy between past and future traffic composition). The precision with which a change point can be located also depends on the length of the time intervals. Shorter intervals increase the precision. Unfortunately, they also increase the probability that small traffic variations (e.g., bursts) are misinterpreted as a change point. This could introduce unwanted noise into the subsequent model generation process. To find a suitable length for the time intervals, we experimented with a variety of values between 20 and 100 seconds. We chose an interval of 50 seconds since it delivered the best results in our tests.

4.2.2 Extracting Model Generation Data

We assume that each change point indicates the time when a bot has received a command and initiated the corresponding response. Based on this assumption, we leverage change points to extract two pieces of information that are needed for the subsequent model generation step.

First, we extract a snippet of the traffic that is likely to contain the command that is responsible for the observed change. Clearly, the snippet contains the traffic within the time interval where the change point is located. Moreover, we take the first 10 seconds of the following interval. The reason is that when a change point occurs very close to the boundary between two intervals, the CPD algorithm might select the wrong one. To compensate for this potential imprecision, the start of the subsequent traffic interval is included. Finally, we include the last 30 seconds of the previous interval, in order to cover typical command response delays. As a result, each snippet contains 90 seconds of network traffic.

The second piece of information required for creating a detection model is a description of the bot response behavior. To this end, we extract a response *behavior profile*. The behavior profile captures the network-level activities of the bot once a command is received. This profile is computed by taking the average of the traffic profile vectors over the complete period where the bot carries out its response. This period is considered to be the time from the start of the current response to the next change in behavior. That is, once the network traffic changes again, we assume that the bot has finished its task or received another command.

4.3 Generating Detection Models

Given a set of network traffic snippets, together with their associated response behavior profiles, we automatically generate suitable detection models. Recall that detection models should embody the correlation of two events : The first event is the appearance of a command in the network traffic. The second event is the appearance of a subsequent response. The patterns that each of the two events have to match are represented separately in our model.

At this point, the set of snippets contains a mix of network traffic that consists of different commands and some contents that are specific to the C&C protocol. For subsequent processing performed by the token extraction algorithm, we require a two-phase clustering : First, we arrange snippets such that those are put together in a *cluster* that likely contain the *same* command. Afterwards, we group the contents of the snippets in each cluster such that elements in a group share commonalities that can be leveraged by the token extraction algorithm (described in Section 4.3.1).

To cluster similar snippets (the first step), we make the following assumption : The network traffic of a bot responding to a certain command will look similar to the traffic generated by this bot executing the *same* command at some later time. On the other hand, the same bot executing a different command will generate traffic that looks different. That is, there is a correspondence between the command that is sent and the response that is invoked. This assumption can be leveraged by clustering the snippets according to the behaviors that we believe to be a response. That is, the goal is to find *behavior clusters*, where each such cluster represents a certain kind of bot activity, such as a scanning period, or a denial of service attack, or any other kind of distinguishable network activity. Once such clusters have been found, we can expect that most snippets that are part of the same cluster contain common parts that are either directly responsible for triggering the bot reaction (the command itself), or at least always have to appear in order for a bot to react that way.

To identify clusters of related activity, we perform hierarchical clustering [42] based on the normalized response behavior profiles. The clustering is stopped when the minimal distance between any two clusters exceeds a threshold of 0.005. This threshold is empirically chosen such that only closely related snippets are clustered together. However, when unrelated snippets end up in the same cluster, the command token extraction algorithm will search for shared strings that do not exist, possibly producing incorrect results.

After the clustering step, each cluster holds a set of snippets that likely contain a command that lead to the same response. In the next step, these snippets are used to extract the model of the bot command (as described in Section 4.3.1). The response behaviors associated with the snippets are then used to model the response activity (as discussed in Section 4.3.2).

4.3.1 Command Model Generation

The objective of the command model generation step is to identify common elements in a set of network snippets that belong to a particular behavior cluster. These common elements are a set of character strings (*tokens*) that appear in the payloads of network connections. In particular, we are interested in finding tokens that appear frequently in the traffic snippets, since there is a chance that they encode bot commands.

To extract likely bot commands from network traces, we use a signature generation technique that produces *token sequences*. A token sequence consists of an ordered set of tokens. That is, the tokens have to appear in a certain order, but there can be arbitrary characters between each token. Token sequences can be easily encoded as regular expressions (which can serve directly as input to a network intrusion detection system).

To find common tokens in a set of network snippets, we use the longest common subsequence algorithm (based on suffix arrays). Unfortunately, the algorithm outputs a token sequence only if this sequence is present in all network traces. Thus, we cannot apply the algorithm directly. The reason is that we cannot require that all traffic snippets contain the same command. For example, it is possible that different commands lead to a similar response, and hence, the corresponding snippets end up in the same cluster. Another problem might be due to an incorrectly detected change point, which can cause an unrelated snippet to become part of a cluster. Therefore, we require a second clustering refinement step that groups similar network packet payloads within each behavior cluster (i.e., snippets that lead to similar response behavior).

For the second clustering step, we employ a standard complete-link, hierarchical clustering algorithm to find payloads that are similar. The algorithm starts by putting each payload into a separate cluster. Then, a matrix is constructed that stores the similarity between each pair of clusters. We define the similarity S between two payloads P_1 and P_2 that share n common tokens $t_1, \ldots, t_n$ as :

$$S = \sum_{i=1}^{n} \left(\frac{len(t_i)}{len(P_1)} + \frac{len(t_i)}{len(P_2)} \right) /2 \tag{4.2}$$

Once the similarity matrix is computed, the algorithm switches to the merging phase. During this phase, similar clusters are successively merged into larger clusters. Two clusters are merged only if the payloads in the clusters have a similarity score greater than a minimum threshold δ. The value for δ is the average of the similarity scores in the similarity matrix.

The longest common subsequence algorithm is applied to each set of similar payloads, generating one token sequence per set. Recall that the second clustering step is performed individually for each behavior cluster. Thus, it is possible (and common) that multiple token sequences are associated with a single behavior cluster. Each of these token sequences represents a potential command that leads to network activity that the corresponding response behavior profile captures.

Some of the generated token sequences may be overly generic. That is, they are likely to match on benign traffic frequently. To improve the precision of our detection models, these token sequences should be identified and removed. This can be done in an automated fashion by matching all generated token sequences against network traffic that is known to be benign. Every match of a token sequence on the benign traffic is clearly undesirable, and thus, suggests to discard the token sequence. To identify as many of the false positive candidates as possible, the benign traffic pool should cover a broad range of protocols and transmitted content. For our experiments, we recorded the traffic at the Secure Systems Lab in Vienna for a duration of one day. The network is well administrated and used exclusively by computer science personnel that is generally security-aware. It is thus safe to assume that all traffic is benign. In addition to removing token sequences that match benign traffic, we remove all token sequences whose longest token is shorter than five bytes. This is done because token sequences consisting only of very short tokens will trigger frequently just by chance, e.g., when large amounts of binary or encrypted data are transmitted.

4.3.2 Response Model Generation

The second part of our detection model consists of a network-based description of the bot response. This description should capture the kind of network activity that is expected to be shown by a bot after the command has been received.

The input to this step is a behavior cluster. Recall that a behavior cluster is created by grouping similar response behavior profiles and their associated snippets. We generate the bot response model for a behavior cluster by computing the element-wise average of the (vectors of the) individual behavior profiles. The result is another behavior profile vector that captures the aggregate of the behaviors combined in the respective behavior cluster. As such, this behavior profile is suitable to model the expected bot response behavior associated with the bot commands that are described by the content-based models extracted from the snippets.

In the previous section, we have mentioned that each token sequence must contain tokens of a minimal length (five bytes) to become part of a detection model, otherwise it might trigger too frequently on legitimate network traffic. A similar consideration applies to response models. More precisely, in some cases, the behavior profile of a bot response can be exceeded by sending a few HTTP packets or by contacting two other hosts. Clearly, such traffic is easily produced by regular users (e.g., surfing the web or using an instant messaging client). Thus, we introduce minimal bounds for certain network features. In particular, we define a threshold of 1,000 for the number of UDP packets that are sent within one time interval (50 seconds), 100 for HTTP packets, 10 for SMTP packets, and 20 for the number of different IPs. When a response profile exceeds *none* of these thresholds, the corresponding behavior cluster (and its token sequences) are not used to generate a detection model. This technique removes a small number of weak profiles that could potentially result in a large number of false positives.

4.3.3 Mapping Models into Bro Signatures

Bro is a network intrusion detection system designed to passively monitor network activity for suspicious or irregular events [77]. One of its key features is the integrated policy and signature scripting language, which enables custom rules for intrusion detection. Due to its flexibility, Bro is an appropriate platform to implement our detection models.

To map a detection model into a Bro specification, we have to encode the model's set of token sequences as well as its behavior profile. For each token sequence, one Bro signature is generated. The signature consists of the concatenation of the individual tokens of a token sequence, using the '.*' regular expression operator. That is, the sequence of the three tokens "this", "is", and "blue" would be represented as the regular expression ".*this.*is.*blue.*". Also, each signature is restricted to match only on inbound or outbound traffic, depending on the bot traffic it had been generated from.

When a token sequence matches, the corresponding detection model is advanced to the second state. At this point, Bro starts to record the traffic of the host that triggered a signature. This is done for a duration of 50 seconds. Then, the system creates a profile from the recorded traffic, using the following four features : number of UDP packets, number of HTTP packets, number of SMTP packets, and number of unique IP addresses. When the observed traffic exceeds, for at least one of these four features, the corresponding value stored in the response profile, we consider this a match. In that case, the host is considered to be bot-infected, and an alert is raised.

4.4 Evaluation

The purpose of the evaluation is to demonstrate that our system generates detection models that are capable of detecting bot-infected hosts with a low false positive rate. In this section, first we will explain how we captured the bot traffic that we analyzed for extracting our botnet detection models. Afterwards, we will present our results and compare them with the most relevant network-based botnet detection system (i.e. BotHunter). We will conclude the section with showing some example signatures our system produced.

4.4.1 Capturing Bot Traffic

When recording bot traffic traces, it is difficult to predict the required amount of time to obtain a representative collection of bot commands and according responses, since it depends on the degree and kind of activity of the botmaster during the observation period. In our experiments, we decided to aim for a capture period of five days. This ensures that we have a good chance of observing a large variety of different commands. However, most bots receive (common) commands after a short time, often within minutes. Thus, we can start to produce a first set of detection models quickly. Then, we wait for several days to capture less frequent commands as well.

Because of the long runtime of the bots as well as our desire to collect as many traces as possible, an important goal when designing the execution environment was to support as many parallel bot instances as possible. We set up a VMware environment on a server with two Intel Xeon 1.86GHz Quad-core processors, 8 GB of memory, and 300 GB of Raid5 disk space. Each VM is running a fully-patched instance of Windows XP with service pack 2, and is able to run with as little as 64 MB main memory. Using this setup, we are able to simultaneously run up to 50 virtual machine instances on our server.

Each of the guest virtual machines is assigned a static, public IP address, and infected with one bot. All network traffic is captured on the host. Since there are no other applications that run and generate network traffic, the bot accounts for all observed network traffic under its host VM's IP address. Of course, a bot requires Internet connectivity, so that it can connect to the command and control infrastructure and receive commands from the botmaster. However, at the same time, we do not wish the bots that we are analyzing to engage in serious and destructive malicious activity such as denial of service attacks. Thus, we have a firewall that rate-limits all outbound network traffic. After each five days capturing period, all VMs are deleted and recreated in a clean state, before the next set of bot samples is executed.

The fact that we use VMware to execute bots could be considered a potential limitation. It is well-known that VMware is easy to fingerprint, and we are aware that a bot could detect our system. However, the problem of VMware detection is not a conceptual limitation of our approach. In future versions, we could use another virtual machine or run the bots directly on real hardware.

We collected a set of 416 different (based on MD5 hash) bot samples. We obtained these malware programs through Anubis, a public malware analysis service [19]. Thus, the samples originate from a wide range of sources and include bots manually submitted by users, binaries collected with the help of honeypots and spam traps, as well as contributions from malware analysis organizations (such as `ShadowServer.org`). The collection period was more than 8 months. All bot samples were executed in our traffic capturing environment, each producing a traffic trace with a length of five days.

4.4.2 Generating Signatures

The bot traffic traces that were collected were divided into families of bots. This was a manual process, based on the content of the traces. However, this step could be automated in the future [17,20]. The classification process yielded a total of 16 different IRC bot families (with 356 traffic traces) and one HTTP bot family consisting of samples of Kraken (also known as Bobax, with 60 traffic traces). In addition, we obtained 30 network captures for the Storm Worm (also known as Peacomm and Zhelatin), which is the most well-known example of a botnet that uses a peer-to-peer protocol for its C&C communication [50]. The Storm Worm captures were separately generated at the University of Mannheim. Thus, in total, there were 446 network traces available as input for our detection model generation process.

Bot family	#DM	#TS	Bot family	#DM	#TS
IRC1	4	57	IRC2	9	50
IRC3	2	11	IRC4	4	94
IRC5	1	8	IRC6	1	20
IRC7	8	53	IRC8	3	72
IRC9	3	17	IRC10	2	7
IRC11	11	35	IRC12	7	21
IRC13	2	8	IRC14	5	38
IRC15	3	24	IRC16	1	1
HTTP	2	5	STORM	2	110
			TOTAL	70	631

Table *4.2: Number of detection models (DM) and token sequences (TS) for each bot family.*

Using these 446 network traces, our system produced a total of 70 detection models. A more precise breakdown of this number for the different bot families is shown in Table 4.2. The table also shows the numbers of token sequences produced. Recall from Section 4.3.1 that there can be multiple token sequences associated with a single detection model, but it is sufficient that a single one triggers to switch the model into the second state (where it checks for suspicious response activity). As can be seen, our system succeeded in producing at least one detection model for each bot family. This is particularly interesting when considering that Storm (Peacomm) uses encrypted commands. When examining the Storm signatures, we observed that our system correctly identified that the byte string "`.mpg;size=`" is characteristic for this bot type. That is, even though we cannot precisely identify a command in the network trace, our algorithm is able to extract specific artifacts of the bot communication. Also, it should be noted that this automatically-generated token sequence is very close to the human-specified signature in Snort [88], a popular network intrusion detection system.

To understand the quality of our automatically-generated detection models, we compared them to the human-developed bot and C&C signatures used by Snort. This serves as an initial, qualitative assessment to determine whether the signatures are "reasonable" and match traffic that a human analyst would associate with bot activity. In many cases, we found that the signatures were very similar to the human-created references, which confirms that our approach is capable of delivering intuitively correct results. This was true for signatures for all three bot classes (IRC, HTTP, and P2P) that we examined. In other cases, we found that our signatures were overly specific, and contained artifacts of a particular bot that was analyzed (e.g., IRC channel names, IP addresses, time stamps). However, it is typically not problematic to include such specific signatures. While they likely do not detect any bots, they typically do not contribute to false alarms either.

```
signature irc1-000-2 {
  dst-ip == local_nets
  payload /.* PRIVMSG #.* :\.asc .*5 0 .*/
}

#DIFFERENT IPS > 20
```

FIGURE *4.1: Automatically-generated Bro signature and corresponding behavior profile for an IRC bot.*

An example of an automatically-generated detection model for a family of IRC bots is shown in Figure 4.1. The token sequence consists of three tokens that need to be identified in an inbound IP packet. The first token (`PRIVMSG #`) contains a part of the IRC protocol header for transmitting a message. This token restricts the signature to match only on IRC traffic. The second token (`:.asc`) contains the command that instructs the receiving bot to begin scanning. The third token (`5 0`) contains parameters for the scan command. At first, it might seem that this token makes the signature overly restrictive. However, very often, the same set of parameters is used for a command. Thus, this is not a significant restriction. In comparison, a human-created Snort signature matches on "`PRIVMSG .*:.*asc`". The network behavior that needs to be matched in the second detection phase (once the token sequence has been identified in the traffic) requires that a host contacts more than 20 distinct IPs within a time period of 50 seconds. This reflects the scan that a bot initiates when receiving the `.asc` command. Only if this second condition is fulfilled as well, the host is reported as bot-infected.

For additional examples of HTTP and P2P detection models, as well as encrypted C&C channels, the reader is referred to the Appendix.

4.4.3 Detection Capability

To obtain a quantitative measure for the capability of our detection models to identify bot-related traffic, we decided to split our set of 446 network traces into training sets and test sets. Each training set contained 25% of one bot family's traces, while the corresponding test set contained the remaining ones. We used the training sets to generate a new set of detection models. Then, this new set of models was loaded into Bro, and we analyzed the traffic traces in the test sets. In total, this procedure was performed four times per family (four-fold cross validation).

Our system reported a bot infection for 88% of the analyzed traces. The remaining 12% of traces did not trigger even a token sequence match. For all traces that did lead to at least one token sequence match, the behavior profile matching phase triggered as well, thus, correctly confirming the bot infection.

To further put the detection results into context, we decided to perform a comparison between our system and BotHunter [52]. BotHunter is the current state-of-the-art tool for detecting individual bot infections. The system uses a number of phases that model different aspects of the bot life cycle (such as spreading, C&C, and malicious activity). To detect bot commands, BotHunter relies on manually-developed signatures (mainly the database of Snort and some custom signatures). To determine the performance of BotHunter, we ran its latest version (v1.0.2, with default settings) on all 446 bot traffic traces. BotHunter identified signs of bot infections for 69% of the traces. The automatically generated signatures produced by our system thus outperform BotHunter by nearly 20%.

	IP space	Avg. pkts/h	Days	IPs flagged	Total alerts	Alerts/day
Aachen	2,048	40M	55	0	0	0
Greece	4,096	17M	102	11	60	0.59
BotHunter	4,096	17M	6	60	5,849	974.34
BotHunter w/o Blist	4,096	17M	6	5	60	10.00

TABLE *4.3: Results from real-world deployments.*

4.4.4 Real-World Deployment

To analyze the amount of false positives that our detection models generate, we extensively evaluated our system in two real-world network environments. More precisely, we deployed one Bro sensor with our detection models in front of the residential homes of RWTH Aachen University and one sensor at a Greek university network. In Aachen, our system monitored a densely-populated /21 network (2K IPs) for a duration of 55 days. In Greece, we monitored a medium-populated /20 network (4K IPs) for 102 days. On average, we observed about 40 million packets per hour in Aachen, while the number in Greece was about 17 million packets. Thus, our experimental evaluation comprises the analysis of traffic in the order of 94 billion network packets over a period of over three months at two different sites in Europe.

The results of our evaluation are summarized in Table 4.3. Our deployment in Aachen yielded no alerts at all over a duration of two months. There were 130 token sequence matches, which were all correctly invalidated by the behavior profile matching phase. This demonstrates the importance of the second phase of our detection models : Random token sequence matches do not lead to an alert, because without the expected bot response, the behavior profile will not be matched.

In the Greek network, our system raised only few alerts, and over a period of over three months, reported a total of 11 hosts (IPs) as bot-infected. These 11 hosts were responsible for 60 alerts. To verify whether these alerts are false positives or indications of true bot infections, we performed manual analysis of the traffic that caused the alarms. In most cases, this led us to the conclusion that an alarm was a false positive. This is also supported by the fact that both networks are well-maintained and bot infections are very rare. However, a definite decision is difficult to make, since we did not have access to the actual hosts.

Typically, all machines that are reported as bot infected must be manually inspected. Thus, it is important that the system does not overload the administrator with incorrect warnings. Considering the average number of alerts per day that our system reports as well as the number of reported IP addresses (shown in Table 4.3), we believe that this goal is clearly met.

Again, in order to set our results in relation with the current state-of-the-art BotHunter, we deployed a BotHunter sensor in the Greek network (we did not obtain permission to install such a sensor in Aachen). Unfortunately, due to performance limitations, we could run either BotHunter or our system on the machine that was provided to us, but not both systems at the same time. Thus, we deployed BotHunter for a period of only six days. Nevertheless, we feel that this period is sufficiently long to draw meaningful conclusions.

The comparison with BotHunter is instructive. We can see that an off-the-shelf BotHunter installation reports almost one thousand alerts per day. Within a period of only six days, 60 different IP addresses are reported as bot infected, each of which would require manual inspection. Given this very high number of false alerts, we investigated the reasons and even attempted to tweak BotHunter to improve its performance. On closer inspection of the alerts, we observed that a significant amount of them are due to two components (phases). These rely on blacklists of known DNS names and IP addresses that are related to malware domains and C&C servers. In an attempt to reduce the amount of BotHunter's false positives, we disabled these two components. An accordingly modified Bothunter setup produced only 10 alerts per day, reporting a total of 5 IP addresses as bot infected during the six day period. While, in contrast to the off-the-shelf setup, the amount of alerts is now manageable by a human administrator, BotHunter still does not reach the low number of false alerts our system generates.

Additionally, disabling the two components that are responsible for the vast majority of false alerts has a significant negative impact on BotHunter's detection capabilities. When rerunning the experiments on the bot traces using the modified version of BotHunter, the number of bots that BotHunter detects drops to only 39%.

Finally, a large fraction (89%) of the alerts raised by our system in the real-world deployments were triggered by only three different detection models. The situation is different for BotHunter : We observed 155 different matching BotHunter C&C signatures during the evaluation in the Greek network. This large diversity of matching signatures makes it difficult to disable a few BotHunter models that are responsible for the bulk of false positives.

A summary of the results of our evaluation are presented in Table 4.4. Our automatically generated detection models clearly outperform the state-of-the-art solution for single bot detection, BotHunter, which relies on sig-

	Our detection models	BotHunter
Detection rate on bot traces	88%	69%
Incorrectly detected IPs in real-world traffic	11	60

TABLE *4.4: Comparison of the detection performance of our detection models vs. BotHunter.*

natures hand-crafted by human experts.

4.4.5 Examples And Comparison To Hand-tuned Signatures

In the following, we present examples of automatically-generated detection models and compare them to human-generated signatures.

```
signature http-000-2 {
  src-ip == local_nets
  payload /.*GET \/reg\?u=1.*&v=187&s=.*&su=.*p=0&e=0&o=0&a=0&wr=75
           HTTP\/1\.1 \x0d\x0aUser-Agent: Mozilla\/4\.0
              \(compatible; MSIE 6\.0; Windows NT 5\.1\)
   \x0d\x0aHost: .*/
}

#HTTP PACKETS > 440
```

FIGURE *4.2: Automatically-generated Bro signature and corresponding behavior profile for Kraken.*

An example signature for a family of HTTP bots is shown in Figure 4.2. In contrast to IRC bots, where the botmaster *pushes* the command to the bots, an HTTP bot periodically queries the command & control server and *pulls* commands. The token sequence can recognize the characteristic HTTP request issued by Kraken bots. Since the request to (and not the response from) the C&C server is searched for, the signature matches on outbound traffic. The corresponding hand-tuned Snort signature is shown in Figure 4.3.

Both signatures should match on (almost) the same set of bot requests. Note that our signature contains some additional parts that belong to the HTTP protocol. Interestingly, these artifacts implicitly encode that the signature is supposed to match in the context of an HTTP request, a fact that needs to be explicitly encoded in the Snort signature. Also, our signature contains more specific information about the typical requests issued by a bot, e.g., that certain variables are always set to 0 in a request. Moreover, the behavior profile requires to see at least 440 HTTP packets within the

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
  (msg: "E4[rb] BLEEDING-EDGE BOTNET HTTP Botnet reg";
  flow: established;uricontent:"/reg?u="; nocase;
  content:"&v="; nocase; within: 15; content:"&s=";
  nocase; within: 15; content:"&su="; nocase; within: 15;
  content:"&p="; nocase; within: 15; classtype: trojan-activity;
  reference:url,www.honeynet.org/papers/bots;
  sid:2001899; rev:7; )
```

FIGURE *4.3: Hand-tuned Snort signature for a family of HTTP bots.*

observation period of 50 seconds.

```
signature storm_u2-000-12 {
  src-ip == local_nets
  payload /.*\.mpg;size=.*/
}

#DIFFERENT IPS > 438 || #UDP PACKETS > 7445 || #SMTP PACKETS > 1720
```

FIGURE *4.4: Automatically-generated Bro signature and corresponding behavior profile for the Storm Worm.*

The system was also successful in producing a signature for the Storm Worm (for this signature, please refer to Figure 4.4). Generating signatures for this bot is a challenging task. First, the commands are not sent by a central server as with IRC and HTTP bots, but the communication infrastructure relies on a peer-to-peer network. Second, the botnet uses an authentication scheme which is XOR-encrypted and afterwards, all communication between the peers is compressed with `zlib`, a data compression library. Thus, our approach of detecting fixed tokens, which represent commands in the network stream, seems to fail : Due to encryption and compression, there are actually no fixed commands we can detect. Nevertheless, our system generated a token sequence that is also very close to the human-specified signature in Snort (see Figure 4.5).

```
alert udp $HOME_NET 1024:65535 -> $EXTERNAL_NET 1024:65535
  (msg:"E7[rb] BOTHUNTER Storm(Peacomm) Peer Coordination Event
  [SEARCH RESULT]"; content:"|E311|"; depth:5; rawbytes;
  pcre:"/[0-9]+\.mpg\;size\=[0-9]+/x"; rawbytes;
  classtype:bad-unknown; sid:9910013; rev:99;)
```

FIGURE *4.5: Hand-tuned Snort signature for the Storm Worm.*

Our algorithm correctly identifies that the byte string `.mpg;size=` is characteristic for this bot type and the Snort signature also uses this specific token sequence for detecting Storm. Thus, our signature captures the same properties of the communication phase, and the detection accuracy is comparable to a human-generated signature. Even if we cannot precisely identify a command in the network trace, our algorithm is able to extract characteristic artifacts of the bot communication. Furthermore, this signature confirms that we do not depend on knowledge about the actual communication protocol of the bot, but can generate signatures for arbitrary botnets.

The behavior profile reflects the participation in the P2P network, which comes with a high number of contacted IP addresses as well as with a high number of UDP packets. Also, it captures the fact that Storm is frequently used for sending spam mails. The profile requires to observe within a time period of 50 seconds more than 438 different contacted IPs, or more than 7,445 UDP packets, or more than 1,720 SMTP packets.

```
signature irc6-003-0 {
  dst-ip == local_nets
  payload /.*:x\.hub\.x 332 .*##russia## :  =PIzNr8s
      Xrm9CDF43axRn\+tNXcpoePVsw0p3KTBdze\+6iX7Af3qMy34Y7W5pm.*/
}

#DIFFERENT IPS > 27
```

FIGURE *4.6: Automatically-generated detection model for an IRC bot using encrypted communication.*

Moreover, we have generated a number of signatures for bots that use encrypted and obfuscated commands over IRC. An example is shown in Figure 4.6. Instead of using plain IRC, these botnets send the commands over IRC in an encrypted form. That is, the botmaster encrypts the command using a specific key and sends it to the IRC channel. Each bot has the decryption key embedded in the binary, decrypts the command, and then executes it.

The token sequence contains botnet specific IRC header artifacts like the channel name and a part of the user name. Also it contains the byte sequence `'332'`, which indicates that an IRC topic message should be matched. The command itself is not intelligible for a human, however, the behavior profile which requires the observation of at least 27 different IP addresses points toward a scan.

Note that encrypted IRC botnets prohibit the use of gray-box testing approaches to trigger responses (such as those introduced by Rajab et al. [83]). Since we have no knowledge about the decryption key, it is very unlikely that we can send valid command tokens. Therefore, it is necessary to monitor the bot in a controlled environment and observe the actual commands sent by the botmaster.

4.5 Limitations

Although our current system is able to effectively detect real-world botnets, we note that it has several limitations. In this section, we discuss the limitations and present possible solutions.

To evade detection, a botmaster may instruct his bots to wait for a certain amount of time before reacting to the command (i.e., he might launch a threshold attack [91]). As a result, our analysis could miss the connection between a command and the appropriate response, both when generating detection models or once the models are deployed. Many other comparable systems rely on a time window of some sort, and thus, are *vulnerable to this same attack* [51–53, 86]. A possible way of handling this evasion attempt is to randomize the time window, making it harder for the adversary to select an appropriate delay. Also, long time delays reduce the usefulness of botnets and increase the difficulty for the attacker [53, 91].

Another limitation of our current implementation is that it uses content-based analysis to detect command tokens. Thus, the system has problems with encrypted command channels. This is a limitation that our approach shares with *all previous techniques* that aim to detect single bots [26,49,52]. To avoid this problem, the most promising approach is to use network-level properties to recognize commands. Interestingly, even in the current version, our system can sometimes identify artifacts that are present in encrypted traffic (as presented in the Appendix). The best example is Storm, for which our system extracts a "command" token that is characteristic for this bot. Also, our system is resistant to simple obfuscation schemes in which a human-readable command is mapped to some unintelligible string. In fact, we have generated token sequences for IRC bot families that match obfuscated commands. This is different from previous approaches, such as BotHunter [52], that deploy manually-developed signatures and thus, can be thwarted by bots that use non-standard commands.

Chapitre 5

Botnet Detection Through Passive DNS Analysis

The domain name service (DNS) plays an important role in the operation of the Internet, providing a two-way mapping between domain names and their numerical identifiers. Given its fundamental role, it is not surprising that a wide variety of malicious activities involve the domain name service in one way or another. For example, bots resolve DNS names to locate their command and control servers, and spam mails contain URLs that link to domains that resolve to scam servers. Thus, it seems beneficial to monitor the use of the DNS system for signs that indicate that a certain name is used as part of a malicious operation.

In this chapter, we introduce EXPOSURE, a system that employs large-scale, passive DNS analysis techniques to detect domains that are involved in malicious activity. We use 15 features that we extract from the DNS traffic that allow us to characterize different properties of DNS names and the ways that they are queried.

Our experiments with a large, real-world data set consisting of 100 billion DNS requests, and a real-life deployment for two weeks in an ISP show that our approach is scalable and that we are able to automatically identify unknown malicious domains that are misused in a variety of malicious activity (such as for botnet command and control, spamming, and phishing).

5.1 Overview

The goal of EXPOSURE is to detect malicious domains that are used as part of malicious operations on the Internet. To this end, we perform a passive analysis of the DNS traffic that we have at our disposal. Since the traffic we monitor is generated by real users, we assume that some of these users are infected with malicious content, and that some malware components will be running on their systems. These components are likely to contact the domains that are found to be malicious by various sources such as public malware domain lists and spam blacklists. Hence, by studying the DNS behavior of known malicious and benign domains, our goal was to identify distinguishable generic features that are able to define the maliciousness of a given domain.

5.1.1 Extracting DNS Features for Detection

Clearly, to be able to identify DNS features that allow us to distinguish between benign and malicious domains, and that allow a classifier to work well in practice, large amounts of training data are required. As the offline dataset, we recorded the recursive DNS (i.e., RDNS) traffic from Security Information Exchange (SIE) [7]). We performed offline analysis on this data and used it to determine DNS features that can be used to distinguish malicious DNS features from benign ones. The part of the RDNS traffic we used as initial input to our system consisted of the DNS answers returned from the authoritative DNS servers to the RDNS servers. An RDNS answer consists of the name of the domain queried, the time the query is issued, the duration the answer is required to be cached (i.e., TTL) and the list of IP addresses that are associated with the queried domain. Note that the RDNS servers do not share the information of the DNS query source (i.e. the IP address of the user that issues the query) due to privacy concerns.

By studying large amounts of DNS data, we defined 15 different features that we use in the detection of malicious domains. 6 of these features have been used in previous research(e.g., [76, 78, 100]), in particular in detecting malicious Fast-Flux services or in classifying malicious URLs [68]. The features that we use in the detection and our rationale for selecting these features are explained in detail in Section 6.2.

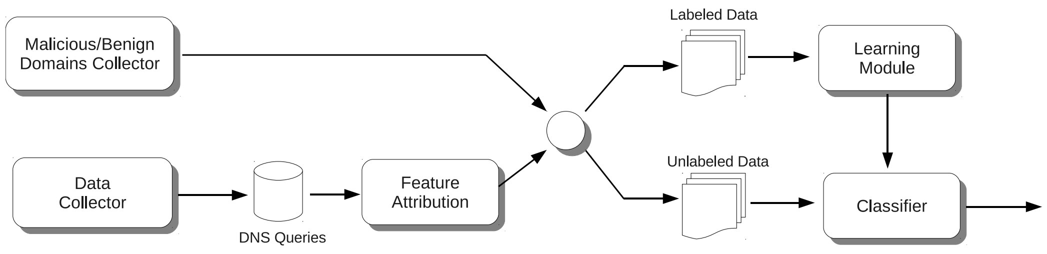


FIGURE *5.1: Overview of* EXPOSURE

5.1.2 Architecture of EXPOSURE

Figure 5.1 gives an overview of the system architecture of the EXPOSURE. The system consists of five main components :

The first component, the Data Collector, records the DNS traffic produced by the network that is being monitored.

The second component is the Feature Attribution component. This component is responsible for attributing the domains that are recorded to the database with the features that we are looking for in the DNS traffic.

The third component, the Malicious and Benign Domains Collector, works independently of, and in parallel to the Data Collector Module. It collects domains that are known to be benign or malicious from various sources. Our benign domains sets are composed of information acquired from Alexa [4] and a number of servers that provide detailed WHOIS [2] data. In contrast, the malicious domain set is constructed from domains that have been reported to have been involved in malicious activities such as phishing, spamming, and botnet infections by external sources such as malwaredomains.com, Phishtank ([79]), and malware analyzers such as Anubis [21]). Note that these lists are constantly updated, and become even more comprehensive over time. The output of the Malicious and Benign Domains Collector is used to label the output of the Feature Attribution component.

Once the data is labeled, the labeled set is fed into the fourth component : The Learning Module. This module trains the labeled set to build malicious domain detection models. Consequently, these models, and the unlabeled domains, become an input to the fifth component : The Classifier.

The Classifier component takes decisions according to the detection models produced by the Learning component so that the unlabeled domains are grouped into two classes : domains that are malicious, and those that are benign.

5.1.3 Real-Time Deployment

The deployment phase of EXPOSURE consists of two steps. In the first step, the features that we are interested in are monitored and the classifier is trained based on a set of domains that are known to be benign or malicious. In a second step, after the classifier has been trained, the detection starts and domains that are determined to be suspicious are reported. Note that after an initial period of seven days of training [1], the classifier is retrained every day. Hence, EXPOSURE can constantly keep up with the behavior of new malware.

5.2 Feature Selection

To determine the DNS features that are indicative of malicious behavior, we tracked and studied the DNS usage of several thousand well-known benign and malicious domains for a period of several months (we obtained these domains from the sources described in Section 5.3). After this analysis period, we identified 15 features that are able to characterize malicious DNS usage. Table5.1 gives an overview of the components of the DNS requests that we analyzed (i.e., feature sets) and the features that we identified. In the following sections, we describe these features and explain why we believe that they may be indicative of malicious behavior.

5.2.1 Time-Based Features

The first component of a DNS record that we analyze is the time at which the request is made. Clearly, the time of an individual request is not very useful by itself. However, when we analyze many requests to a particular domain over time, patterns indicative of malicious behavior may emerge. In particular, we examine the changes of the volume (i.e., number) of requests for a domain. The time-based features that we use in our analysis are novel and have not been studied before in previous approaches.

One of our insights is that malicious domains will often show a sudden increase followed by a sudden decrease in the number of requests. This is because malicious services often use a technique called *domain flux* [93]

1. We have experimentally determined the optimal training period to be seven days (see Section 5.3.2.)

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

Table *5.1: Features.(LMS = Longest Meaningful Substring)*

to make their infrastructures more robust and flexible against take downs. Each bot may use a domain generation algorithm (DGA) to compute a list of domains to be used as the command and control server or the dropzone. Obviously, all domains that are generated by a DGA have a short life span since they are used only for a limited duration. Examples of malware that make use of such DGAs are Kraken/Bobax [12], the Srizbi bots [107] and the Conficker worm [80]. Similarly, malicious domains that have recently been registered and are involved in scam campaigns will show an abrupt increase in the number of requests as more and more victims access the site in a short period of time.

To analyze the changes in the number of requests for a domain during a given period of time, we divide this period into fixed length intervals. Then, for each interval, we can count the number of DNS queries that are issued for the domain. In other words, the collection of DNS queries that target the domain under analysis can be converted into time series (i.e., chronologically ordered sequences of data values). Hence, we can leverage off-the-shelf algorithms [18]. We perform our time series analysis on two different scopes : First, we analyze the time series globally. That is, the start and end times of the time series are chosen to be the same as the start and the end times of the entire monitoring period. Second, we apply local scope time series analysis where the start times and end times are the first and last time the domain is queried during the analysis interval. While the global scope analysis is used for detecting domains that either have a short life or have changed their behavior for a short duration, the local scope analysis focuses on how domains behave during their life time.

A domain is defined to be a *short-lived domain* (i.e., Feature 1) if it is queried only between time t_0 and t_1, and if this duration is comparably short (e.g., less than several days). A domain that suddenly appears in the global scope time series and disappears after a short period of activity has a fairly abnormal behavior for being classified as a benign domain. Normally, if a domain is benign, even if it is not very popular, our thesis is that the number of queries it receives should exceed the threshold at least several times during the monitoring period (i.e., two and a half months in our experiments). Therefore, its time series analysis will not result in an abrupt increase followed by a decrease as the time series produced by a short-lived domain does.

The main idea behind performing local scope analysis is to zoom into the life time of a domain and study its behavioral characteristics. We mainly focus on three features (i.e., Features 2, 3, 4) that may distinguish malicious and benign behavior either by themselves or when used in conjunction with other features. All the features involve finding similar patterns in the time series of a domain. Feature 2 checks if there are domains that show daily similarities in their request count change over time (i.e., an increase or decrease of the request count at the same intervals everyday). Feature 3 aims to detect regularly repeating patterns. Finally, Feature 4 checks whether the domain is generally in an "idle" state (i.e., the domain is not queried) or is accessed continuously (i.e., a popular domain).

The problem of detecting both short-lived domains and domains that have regularly repeating patterns can be treated as a change point detection (CPD) problem. CPD algorithms operate on time series and their goal is to find those points in time at which the data values change abruptly. The CPD algorithm that we implemented [18] outputs the points in time the change is detected and the average behavior for each duration. In the following section, we explain how we interpret the output of the CPD to detect the short-lived domains and the domains with regularly repeating patterns.

Detecting abrupt changes

As CPD algorithms require the input to be in a time series format, for each domain, we prepare a time series representation of their request count change over time. Our interval length for each sampling point is 3600 seconds (i.e., one hour). We chose 3600 seconds as the interval length after experimenting with different values (e.g., 150, 300 etc.).

Before feeding the input directly into the CPD algorithm, we normalize the data with respect to the local maximum. Then, we make use of the well-known CUSUM (cumulative sum) robust CPD algorithm that is known to deliver good results for many application areas [18]. CUSUM is an online algorithm that detects changes as soon as they occur. However, since we record the data to a database before analyzing it, our offline version of the CUSUM algorithm yields even more precise results (i.e., the algorithm knows in advance how the "future" traffic will look like as we have already recorded it).

Our algorithm to identify change points works as follows : First, we iterate over every time interval $t = 3600$ seconds, from the beginning to the end of the time series. For each interval t, we calculate the average request count P_t^- for the previous $\epsilon = 8$ time intervals and the traffic profile P_t^+ for the subsequent ϵ intervals. We chose ϵ to be 8 hours based on the insight that a typical day consists of three important periods : working time, evening and night. Second, we compute the distance $d(t)$ between P_t^- and P_t^+. More precisely :

$$P_t^- = \sum_{i=1}^{\epsilon} \frac{P_{t-i}}{\epsilon} \quad P_t^+ = \sum_{i=1}^{\epsilon} \frac{P_{t+i}}{\epsilon} \quad d(t) = \left|P_t^- - P_t^+\right| \tag{5.1}$$

The ordered sequence of values $d(t)$ forms the input to the CUSUM algorithm. Intuitively, a change point is a time interval t for which $d(t)$ is sufficiently large and is a local maximum.

The CUSUM algorithm requires two parameters. The first parameter is an upper bound ($local\_max$) for the normal, expected deviation of the present (and future) traffic from the past. For each time interval t, CUSUM adds $d(t) - local\_max$ to a cumulative sum S. The second parameter determines the upper bound ($cusum\_max$) that S may reach before a change point is reported. To determine a suitable value for $local\_max$, we require that each individual traffic feature may deviate by at most $allowed\_avg\_dev = 0.1$. Based on this, we can calculate the corresponding value $local\_max = \sqrt{dim \times allowed\_avg\_dev^2}$. Since in our application, there is only one dimension, the $local\_max = allowed\_avg\_dev$. For $cusum\_max$, we use a value of 0.4. Note that we determined the values for *allowed_avg_dev* and *cusum_max* based on empirical experiments and measurements.

The CPD algorithm outputs the average request count for each period a change is detected and the time that the change occurs. Since we employ the CPD algorithm for two purposes (namely to detect short-lived domains and domains that have repeating patterns), we run it twice. We first use the global scope time series and then the local scope time series as input. When the CPD is run with global time series, it can detect short-lived domains. Short-lived domains tend to have two sudden behavioral changes, whereas domains that are continuously queried have multiple change points. On the other hand, to detect the domains with repeating patterns on their local scope time series, we associate the number of the changes and the standard deviation of the durations of the detected changes.

Detecting similar daily behavior

A typical technique to measure the level of similarity of two time series is to calculate the distance between them [57]. To determine whether a domain produces similar time series every day, we calculate the Euclidean Distance between every pair of time series of a domain. Euclidean Distance is a popular distance measuring algorithm that is often used in data mining [22, 101, 113].

We first need to break the local time series produced for each domain into daily time series pieces. Each day starts at 00 :00 am and finishes at 23 :59 pm. Assuming that a domain has been queried n days during our analysis period, and $d_{i,j}$ is the Euclidean Distance between i_{th} day and j_{th} day, the final distance D is calculated as the average of $(n-1)*(n-2)/2$ different distance pairs, as shown in the following formula :

$$D = (\sum_{i=1}^{n} \sum_{j=i+1}^{n} d_{i,j})/((n-1)*(n-2)/2) \tag{5.2}$$

Using the Euclidean Distance, the results are sensitive to small variations in the measurements (e.g., 1000 requests between 9 and 10 am compared to 1002 requests between the same time period may fail to produce a correct similarity result although the difference is not significant). A common technique to increase the correctness of the results is to apply preprocessing algorithms to the time series before calculating the Euclidean Distance [34]. In our preprocessing step, we transform the time series $T = t_1, t_2, ..., t_n$, where n is number of intervals, into two phases. In the first phase, we perform offset translation by subtracting the mean of the series from each value (i.e., $T = T - mean(T)$). In the second phase, we scale the amplitude by dividing each value by the variance (i.e., $T = (T - mean(T))/std(T)$) [34].

5.2.2 DNS Answer-Based Features

The DNS answer that is returned by the server for a domain generally consists of several DNS A records (i.e., mappings from the host to IP addresses). Of course, a domain name can map to multiple IP addresses. In such cases, the DNS server cycles through the different IP addresses in a round robin fashion [1] and returns a different IP mapping each time. This technique is useful in practice for load balancing.

Malicious domains typically resolve to compromised computers that reside in different Autonomous Systems (ASNs), countries, and regions. The attackers are opportunistic, and do not usually target specific countries or IP ranges. Whenever a computer is compromised, it is added as an asset to the collection. Also, attackers typically use domains that map to multiple IP addresses, and IPs might be shared across different domains.

With this insight, we extracted four features from the DNS answer (i.e., feature set F2). The first feature is the number of different IP addresses that are resolved for a given domain during the experiment window (Feature 5). The second feature is the number of different countries that these IP addresses are located in (Feature 6). The third feature is the reverse DNS query results of the returned IP addresses (Feature 7). The fourth feature (Feature 8) is the number of distinct domains that share the IP addresses that resolve to the given domain. Note that Features 5, 6, and 7 have been used in previous work (e.g., [**?**, 14, 78, 100]).

Although uncommon, benign domains may also share the same IP address with many other domains. For example, during our experiments, we saw that one of the IP addresses that belongs to *networksolutions.com* is shared by 10, 837 distinct domains. This behavior is sometimes exhibited by web hosting providers and shared hosting services.

To determine if an IP is used by a shared hosting service, we query Google with the reverse DNS answer of the given IP address. Legitimate web hosting providers and shared hosting services are typically ranked in the top 3 query answers that Google provides. This helps us reduce false positives.

5.2.3 TTL Value-Based Features

Every DNS record has a *Time To Live* (TTL) that specifies how long the corresponding response for a domain should be cached. It is recommended that the TTL is set to between 1 and 5 days so that both the DNS clients and the name servers can benefit from the effects of DNS caching [3].

Systems that aim for high availability often set the TTL values of host names to lower values and use Round-Robin DNS. That is, even if one of the IP addresses is not reachable at a given point in time, since the TTL value expires quickly, another IP address can be provided. A representative example for such systems are Content Delivery Networks (CDNs).

Unfortunately, setting lower TTL values and using Round-Robin DNS is useful for the attackers as well. Using this approach, malicious systems achieve higher availability and become more resistant against DNS blacklisting (DNSBL) [5] and take downs. For example, Fast-Flux Service Networks (FFSN) [100] are malicious systems that abuse Round-Robin DNS.

Most techniques to detect FFSNs are based on analyzing abnormal usage patterns of Round-Robin DNS. More precisely, to label a domain as being a member of an FFSN, previous research [78,100] expects to observe a low TTL usage combined with a constantly growing DNS answers list (i.e., distinct IP addresses).

We extracted five features from the TTL value included in the DNS answers (see Table 5.1). The average TTL usage feature (Feature 9) was introduced in previous research [78]. The rest of the features (i.e., Features 10, 11, 12, 13) have not been used before in previous work.

During our experiments with large volumes of DNS traffic, we observed that frequent TTL changes are exhibited by malicious networks that have a sophisticated infrastructure. In such networks, some of the bots are selected to be proxies behind which other services (e.g., command and control servers) can be hidden. The managers of such malicious networks assign different levels of priorities to the proxy bots by setting lower TTL values to the hosts that are less reliable. For example, there is a good chance that a proxy running on an ADSL line would be less reliable than a proxy running on a server running in a university environment.

To determine the validity of our assumption about this type of TTL behavior, we tracked the Conficker domains for one week. We observed that different TTL values were returned for the IPs associated with the Conficker domains. While the static IP addresses have higher TTL values, the dynamic IP addresses, that are most probably assigned to home computers by Internet service providers, have lower TTL values (e.g., adsl2123-goland.net would have a lower TTL value than a compromised host with the domain name workstation.someuniversity.edu).

We observed that the number of TTL changes and the total number of different TTL values tend to be significantly higher in malicious domains than in benign domains. Also, malicious domains exhibit more scattered usage of TTL values. We saw that the percentage for the usage of some specific ranges of TTL values is often indicative of malicious behavior. Based on our empirical measurements and experimentations, the TTL ranges that we investigate are $[0, 1)$, $[1, 10)$, $[10, 100)$, $[100, 300)$, $[300, 900)$, $[900, inf)$. Malicious domains tend to set their TTL values to lower values compared to benign domains. In particular, the range of $[0, 100)$ exhibits a significant peak for malicious domains.

5.2.4 Domain Name-Based Features

Benign services usually try to choose domain names that can be easily remembered by users. For example, a bank called "The Iceland Bank" might have a domain name such as "www.icelandbank.com". In contrast, attackers are not concerned that their domain names are easy to remember. This is particularly true for domain names that are generated by a DGA.

The main purpose of DNS is to provide human-readable names to users as they often cannot memorize IP addresses of servers. Therefore, benign Internet services tend to choose easy-to-remember domain names. In contrast, having an easy-to-remember domain name is not a concern for people who perform malicious activity. This is particularly true in cases where the domain names are generated by a DGA. To detect such domains, we extracted two features from the domain name itself : First, the ratio of the numerical characters to the length of the domain name (Feature 14), and second, the ratio of the length of the longest meaningful substring (i.e., a word in a dictionary) to the length of the domain name (Feature 15).

Note that there exist popular domains such as *yahoo.com* and *google.com* that do not necessarily include "meaningful" words. In order to gain a higher confidence about a domain, we query Google and check to see if it returns a hit-count for a domain that is above a pre-defined threshold.

When analyzing a domain, we only focus on the second level domains (i.e., SLD). For example, for x.y.server.com, we would take server.com. To detect domain names that have been possibly automatically generated, we calculate the percentage of numerical characters (Feature 14) and the ratio of the length of the longest meaningful substring to the total length of the SLD (Feature 15). To extract all possible meaningful substrings from an

SLD, we check the English dictionary.

As some benign domains in China and Russia consist of combinations of alphabetical and numerical characters, Feature 15 produces a high positive rate. However, when Features 14 and 15 are combined, the false positives decrease. Also, for domains that are determined to be suspicious, we check how many times it is listed by Google. The reasoning here is that sites that are popular and benign will have higher hit counts.

5.3 Building Detection Models

5.3.1 Constructing the Training Set

The quality of the results produced by a machine learning algorithm strongly depends on the quality of the training set [99]. Our goal is to develop a classifier that is able to label domains as being benign, or malicious. Thus, we require a training set that contains a representative sample of benign and malicious domains. To this end, we studied several thousand malicious and benign domains, and used them for constructing our training set.

We collected malicious domains from multiple sources. Specifically, we obtained malicious domains from malwaredomains.com [45], the Zeus Block List [66], Malware Domains List [65], Anubis [21] reports, a list of domains that are extracted from suspected to be malicious URLs analyzed by Wepawet [37], and Phishtank [79]. We also used the list of domains that are generated by the DGAs of the Conficker [80] and Mebroot [93] (i.e., Torpig) botnets. These malicious domain lists represent a wide variety of malicious activity, including botnet command and control servers, drive-by download sites, phishing pages, and scam sites that can be found in spam mails.

Note that we are conservative when constructing the malicious domain list. That is, we apply a preliminary check before labeling a domain as being malicious and using it in our training set. Malicious domain sources such as Wepawet and Phishtank operate on URLs that have been submitted by users. Hence, while most URLs in these repositories are malicious, not all of them are. Also, while some third level domains (3LD) of a domain extracted from a URL may behave maliciously, the rest may not (e.g., a.x.com might be malicious, while x.com might be benign).

Assuming that a domain that is suspected to be malicious either by Wepawet or Phishtank has t_{total} possible 3LDs (number of distinct 3LD recorded by EXPOSURE during the analysis period) and t_{mal} 3LDs are thought to be malicious, we choose the domain to be representative for a malicious behavior only if t_{mal}/t_{total} is greater than 0.75 (i.e., only if 75% of the 3LDs have been reported to be involved in malicious activities). The initial malicious domain list that we generated consists of 3500 domains.

As discussed in detail in Section 5.4.1, we assume that all of the Alexa top 1000 domains and domains that we have observed on our sensors that are older than one year are benign. Therefore, we construct our initial benign domain list using these domains. However, to ensure that our benign domain list does not include any domain that might have been involved in malicious activity, we perform a two-step verification process.

First, we compare all the domains in the benign domain list with the malicious domain list and with the tools that test domains for their maliciousness, specifically with McAffee Site Advisor and Norton Safe Web. Second, we also cross-check the benign domain with the list provided by the Open Directory Project (ODP – a large, human-edited directory of the web constructed and maintained by volunteer editors). Our initial benign domain list consists of 3000 domains.

5.3.2 The Initial Period of Training

By experimenting with different values, we determined that the optimal period of initial training for our system was seven days. This period is mainly required for us to be able to use the time-based features that we described in Section 6.2. During this time, we can observe the time-based behavior of the domains that we monitor and can accurately take decisions on their maliciousness.

After the initial one week of training, we are able to retrain the system every day, hence, increasing detection accuracy.

5.3.3 The Classifier

Our classifier is built as a J48 decision tree algorithm (J48). J48 [106] is an implementation of the C4.5 algorithm [82] that is designed for generating either pruned or unpruned C4.5 decision trees. It constructs a decision tree from a set of labeled training set by using the concept of information entropy (i.e., the attribute values of the training set).

The J48 algorithm leverages the fact that the tree can be split into smaller subtrees with the information obtained from the attribute values. Whenever the algorithm encounters a set of items that can clearly be separated from the other class by a specific attribute, it branches out a new leaf according to the value of the attribute. Each time a decision needs to be taken, the attribute with the highest normalized gain is chosen. Among all possible values of the attributes, if there is any value for which there is no ambiguity, the branch is terminated and the appropriate label is assigned to it. The splitting procedure stops when all instances in all subsets belong to the same class.

We use a decision tree classifier because these algorithms have shown to be efficient while producing accurate results [82]. As the decision tree classifier builds a tree during the training phase, the features that are best in separating the malicious and the benign domains can be clearly seen.

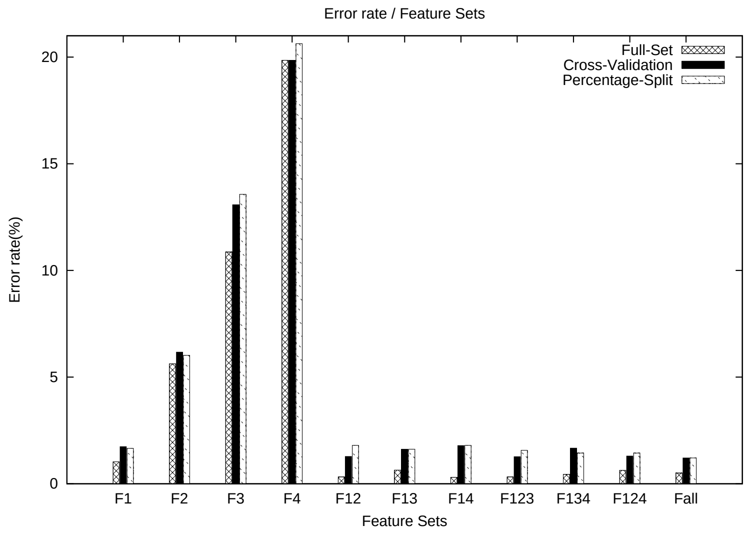


FIGURE *5.2: Percentage of miss-classified instances*

Recall that we divided the 15 features that we use into four different classes according to the type of information used : Features that are extracted from the time series analysis (F1, Time-Based Features), the DNS answer analysis (F2, DNS Answer-Based Features), the TTL value analysis (F3, TTL Value-Based Features), and the analysis of the domain name (F4, Domain Name-Based Features).

To find the combination of features that produce the minimum error rate, we trained classifiers using different combinations of feature sets and compared the results. Figure 5.2 shows the percentage of the number of miss-classified items with three different training schemes : 10-fold cross validation, 66% percentage split, and training on the whole training set. Note that the smallest error rates were produced by F1. Therefore, while experimenting with different combinations of feature sets, we excluded the combinations that do not include F1 (i.e., F23, F24, F34 and F234). The highest error rates are produced by F3 and F4. However, when all features are combined (i.e., F-all), the minimum error rate is produced. Hence, we use the combination of all the features in our system.

5.4 Evaluation

5.4.1 DNS Data Collection for Offline Experiments

Our sensors for the SIE DNS feeds receive a large volume of traffic (1 million queries per minute on average). Therefore, during our offline experimental period of two and a half months, we monitored approximately 100 billion DNS queries. Unfortunately, tracking, recording and post-processing this volume of traffic without applying any filtering was not feasible in practice.

Hence, we reduced the volume of traffic that we wished to analyze to a more manageable size by using two filtering policies. The goal of these policies was to eliminate as many queries as possible that were not relevant for us. However, we also had to make sure that we did not miss relevant, malicious domains.

The first policy we used whitelisted popular, well-known domains that were very unlikely to be malicious. To create this whitelist, we used the Alexa Top 1000 Global Sites [4] list. Our premise was that the most popular 1000 websites on the Internet would not likely be associated with domains that were involved in malicious activity. These sites typically attract many users, and are well-maintained and monitored. Hence, a malicious popular domain cannot hide its malicious activities for long. Therefore, we did not record the queries targeting the domains in this whitelist. The domains in the whitelist received 20 billion queries during two and a half months. By applying this first filtering policy, we were able to reduce 20% of the traffic we were observing.

The second filtering policy targeted domains that were older than one year. The reasoning behind this policy was that many malicious domains are disclosed after a short period of activity, and are blacklisted. As a result, some miscreants have resorted to using domain generation algorithms (DGA) to make it more difficult for the authorities to blacklist their domains. For example, well-known botnets such as Mebroot [93] and Conficker [80] deploy such algorithms for connecting to their command and control servers. Typically, the domains that are generated by DGAs and registered by the attackers are new domains that are at most several months old. In our data set, we found 45.000 domains that were older than one year. These domains received 40 billion queries. Hence, the second filtering policy reduced 50% of the remaining traffic, and made it manageable in practice.

Clearly, filtering out domains that do not satisfy our age requirements could mean that we may miss malicious domains for the training that are older than one year. However, our premise is that if a domain is older than one year and has not been detected by any malware analysis tool, it is not likely that the domain serves malicious activity. To verify the correctness of our assumption, we checked if we had filtered out any domains that were suspected to be malicious by malware analysis tools such as Anubis and Wepawet. Furthermore, we also queried reports produced by Alexa [4], McAfee Site Advisor [8], Google Safe Browsing [6] and Norton Safe Web [9]. 40 out of the 45, 000 filtered out domains (i.e., only 0.09%) were reported by these external sources to be "risky" or "shady". We therefore believe that our filtering policy did not miss a significant number of malicious domains because of the pre-filtering we performed during the offline experiments.

5.4.2 Evaluation of the Classifier

To evaluate the accuracy of the J48 DecisionTree Classifier, we classified our training set with 10-fold cross-validation and percentage split, where 66% of the training set is used for training, and the rest is used to check the correctness. Table 5.3 reports the results of the experiment. The Area Under the ROC curve [28] for the classifier is high for both methods.

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

FIGURE *5.3: Classification accuracy. (AUC=Area Under the ROC Curve)*

Note that the false positive rate is low (i.e., around 1% for both methods). After investigating the reasons for the miss-classifications, we saw that the classifier had identified 8 benign domains as being malicious. The reason for the misclassification was that these domains were only requested a small number of times during the two and half months of experiments (i.e., making the classifier conclude that they were short-lived) and because they exhibited

TTL behavior that looked anomalous (e.g., possibly because there was a configuration error, or because the site maintainers were experimenting to determine the optimal TTL value).

5.4.3 Experiments with the Recorded Data Set

During the two and a half month offline experimental period, we recorded and then analyzed 4.8 million distinct domain names that were queried by real Internet users. Note that a domain that only receives a few requests cannot produce a time series that can then be used for the time-based features we are analyzing. This is because a time series analysis produces accurate results only when the sampling count is high enough. In order to find the threshold for the minimum number of queries required for each domain, we trained our known malicious and benign domain list with differing threshold values. Figure 5.4 shows the detection and false positive rates for the threshold values we tested. Based on these empirical results, we set the threshold to 20 queries, and excluded the 4.5 million domains from our experiments that received less than 20 requests in the two and a half months duration of our monitoring.

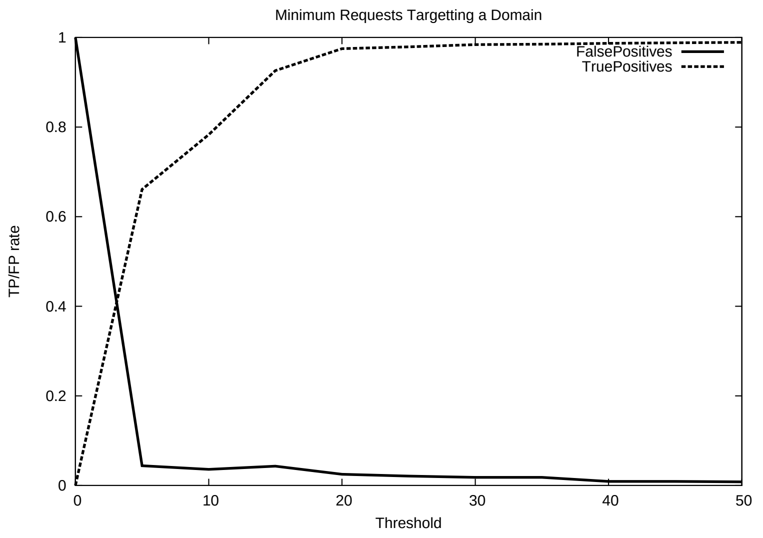


FIGURE *5.4: The effect of the minimum request count on detection rate*

For further experiments, we then focused on the remaining 300,000 domains that were queried more than 20 times. EXPOSURE decided that 17,686 out of the 300,000 domains were malicious (5.9%).

Evaluation of the Detection Rate

The percentage split and cross-validation evaluations on the training set show that the detection rate of our classifier is around 98%. Since our goal is to be able to detect unknown malicious domains that have not been reported by any malicious domain analyzer, our evaluation of the classifier needs to show that we are able to detect malicious domains that do not exist in our training set. To this end, we used malwareurls.com, a malware domains list that we had not used as a source for the initial malicious domains training set.

During the period we performed our experiments, malwareurls.com reported 569 domains as being malicious. Out of these 569 domains, 216 domains were queried by the infected machines in the networks that we were monitoring. The remaining 363 malware domains were not requested. Therefore, in our detection rate evaluation, we take into account only the 216 requested domains.

5 of the 216 domains were queried less than 20 times during entire monitoring period. Since we filter out domains that are requested less than 20 times, we only fed the remaining 211 domains to our system. In the experiments, all of these domains (that were previously unknown to us) were automatically detected as being malicious by EXPOSURE. Hence, the detection rate we observed was similar to the detection rate (i.e. 98%) estimated by the percentage split and cross-validation evaluations on the training set.

Obviously, our approach is not comprehensive and cannot detect all malicious domains on the Internet. However, its ability to detect a high number of unknown malicious domains from DNS traffic is a significant improvement over previous work.

Evaluation of the False Positives

As the domains in our data set are not labeled, determining the real false positive rate is a challenge. Unfortunately, manually checking all 17,686 domains that were identified as being malicious is not feasible. This is because it is difficult, in practice, to determine with certainty (in a limited amount of time) that a domain that is engaged in suspicious behavior is indeed malicious. Nevertheless, we conducted three experiments to make estimates about the false positives of our detection.

In order to obtain more information about the domains in our list, we first tried to automatically categorize them into different groups. For each domain, we started Google searches, checked well-known spamlists, and fed the domains into Norton Safe Web (i.e., Symantec provided us internal access to the information they were collecting about web pages). We divided the domains into ten groups : spam domains (Spam), black-listed domains (BlackList), malicious Fast-Flux domains (FastFlux), domains that are queried by malware that are analyzed by malware analysis tools (Malware), Conficker domains (Conficker), domains that have adult content, domains that are suspected to be risky by Norton Safe Web and McAfee Site Advisor (Risky), phishing domains (Phishing), domains about which we were not able to get any information either from Google or from other sources (No Info), and finally, benign domains that are detected to be malicious (False Positives) (See Table 5.2).

In the first experiment, we manually investigated 50 random malicious domains from our list of 17,686. We queried Google, checked websites that discuss malicious networks, and tried to identify web links that reported a malicious behavior by the domain. Among the 50 randomly chosen domains, the classifier detected three benign domains as being malicious. All these domains had an abnormal TTL change behavior.

In the second experiment, we automatically cross-checked the malicious and suspicious domains that we had identified with our classifier using online site rating tools such as McAfee Site Advisor, [8], Google Safe Browsing [6] and Norton Safe Web [9]. The results show that the false positive estimate is around 7.9% for the malicious domains that we identified.

Note that EXPOSURE did not generate any false positives during the two week real-time, real-world deployment in an ISP as discussed in the next section.

5.4.4 Real-World, Real-Time Detection with EXPOSURE

To test the feasibility and scalability of EXPOSURE as a malicious domain detector in real-life, we deployed it in the network of an ISP that provided us complete access to its DNS servers for two weeks. These servers receive DNS queries from a network that supports approximately 30,000 clients.

During the two-week experimental period, EXPOSURE analyzed and classified 100 million DNS queries. No pre-filtering was applied. At the end of two weeks, EXPOSURE detected 3117 new malicious domains that were previously not known to the system and had not been used in the training. 2821 of these domains fall into the category of domains that are generated by a DGA and all belong to the same malicious entity. 5 out of the remaining 396 domains were reported as being malicious domains by security companies

MW-Group	Rand 50	Malicious
Spam	18	3691
Black-List	8	1734
FastFlux	-	114
Malware	6	979
Conficker	4	3693
Adult	3	1716
Risky	-	788
Phishing	3	0
No Info	5	2854
False Positives	3 (6%)	1408 (7.9%)

Table *5.2: Tests for False Positives*

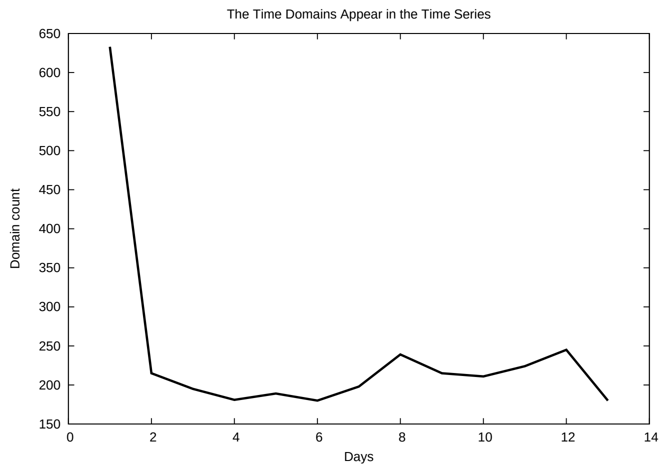
such as Anvira, one month after we had detected them.

We cross-checked the rest of the remaining domains we had detected. All detected domains were classified as being risky by McAfee Site Advisor [8].

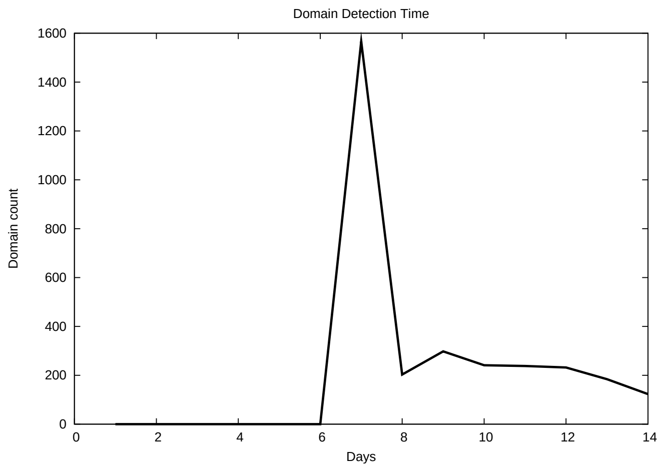
Figures 5.5(a) and 5.7 show the number of new, previously unknown malicious domains detected every day. As can be seen, after the initial seven days of local training in the network being monitored, EXPOSURE started to produce daily detections and detected 200 new malicious domains per day on average.

After the experiments, we provided the ISP with the list of clients that were potentially infected, or had been victims of scams.

The distinct number of IP addresses that queried the malicious domains that EXPOSURE detected were 3451. Since the ISP applies a dynamic IP assignment to its clients, this number does not represent the exact number of infected machines in the network. To estimate the number of infected machines in the network, we grouped the malicious domains according to the IP addresses they are mapped to. There were 5 different groups of malicious domains. We then calculated the average number of distinct IP addresses that issued DNS queries to the domains in these 5 groups every hour. We chose one hour as an interval by assuming that the users in the network stay online at least an hour before they disconnect. Table 5.3 lists the number of clients that attempt to access the domains that fall into the different malware groups. We estimate that there were about 800 machines on the network that issued the requests to the malicious domains.



(a)



(b)

FIGURE *5.5: The first time a domain is queried and the first time it is detected*

5.4.5 Comparison with Previous Work

The Fast-Flux Detectors

The results in Table 5.2 show that 114 of the malicious domains that our classifier has identified during the initial training phase fall in to category

Groups	Avg Life Time	Most frequent life time	# of infected clients
DGA domains	1.2 days	0.99 days	49
Iksmas Worm	11.9 days	11.9 days	70
Worm :Win32/Slenping	12.0 days	12.0 days	253
Trojan-Generic.dx	11.9 days	11.9 days	70
Other	10.8 days	11.9 days	391
Total			833

TABLE *5.3: Information on the detected malicious domains*

of Fast-Flux Service Networks (FFSNs). Since we claim that our approach is able to detect a wide range of malicious domains including FFSNs, we compare our detection rate for this threat with the most recent published work in this area (i.e., [78]).

Perdisci et. al. [78] filters out all of the domains that are not likely to be classified as being FFSN. When we applied the same policy to our two and half month data set, 300,000 domains were filtered out and 5,771 were left as candidates for FFSNs. When we classified these domains with the feature set Perdisci et. al. use in their paper, we detected 114 FFSNs using their approach. Hence, our approach is as good in detecting FFNSs as Perdisci et. al. although it is a much more generic system.

Notos : Reputation-based Malicious Domain Detection

Very recently, Antonakakis et al. [14] concurrently and independently proposed a detection scheme that is similar to our work. The proposed system, Notos, dynamically assigns reputation scores to domain names whose maliciousness has not been discovered yet. A detection scheme is built that is based on the premise that agile malicious uses of DNS have unique characteristics. Hence, the claim is that malicious use of DNS can be distinguished from benign use.

To be able to define these unique characteristics, the authors analyze a number of features that are grouped into three categories : Network-based features, zone-based features (i.e. features that are extracted from the domain name itself, either by string analysis or with the information obtained from whois service) and evidence-based features.

While the network-based features are employed for combing out the domains that do not exhibit fluxy behavior (i.e. stable DNS usage), the zone-based features are used for distinguishing between legitimate CDNs and the domains that are likely to be malicious. After this two-layer classification, reputation scores are given to the domains. In other words, all of the domains and the IP addresses they are mapped to are compared with already known lists of domains or IP addresses that host malicious entities. This third step of classification is done using evidence-base features.

In their paper, as a limitation of Notos, the authors state that Notos is not able to detect malicious domains that are mapped to a new address space each time and never used for other malicious purposes again. This limitation stems from the fact that Notos strongly relies on network-based features. EXPOSURE does not have this limitation as it uses time-based features. Since such domains would have a short life, they would appear in the time series and disappear immediately after they are deactivated by the attacker. Hence, unlike Notos, we are able to detect such domains.

As discussed before, the 2821 automatically generated malicious domains that we detected in the real-life traffic of the ISP had an average lifespan of 1.2 days (see Table 5.3). That is, all these domains were short-lived domains. During their life time, on average, they mapped to 3.14 distinct IP addresses. In the first phase of Notos' detection scheme, the domains are divided into two categories : domains that have a stable network-model and domains that have a non-stable network-model. Since the automatically generated malicious domains that we are able to detect do not use a wide range of IP addresses, Notos might classify these domains as domains with a stable network profile. In other words, we believe that Notos might miss-classify them.

Also, as the authors discuss in their paper, because Notos is a reputation-based system, there may be cases where legitimate domains that are hosted in "bad neighborhoods" may be identified as being malicious. In comparison, the features that EXPOSURE relies on do not cause such false positives as no historical information on IPs or domains are utilized.

One main advantage of EXPOSURE over Notos is that Notos requires a large passive DNS collection and sufficient time to create an accurate, passive DNS database. First, it is unclear how much time is required for this database to be comprehensive. Second, this database needs to be constantly updated with large data-feeds in order to remain accurate and to have a wide overview of malicious activities on the Internet. In comparison, as we show in our evaluation, EXPOSURE only required a week of local training, and much less DNS data in the network of a medium ISP to be able to detect unknown domains.

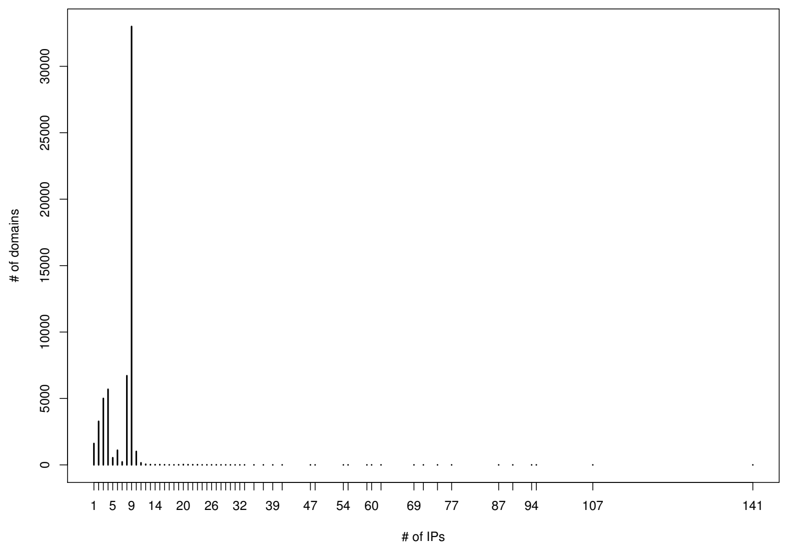


FIGURE *5.6: Number of IP addresses mapped to the malicious domains detected by EXPOSURE*

5.5 Real-World Deployment of EXPOSURE

After we published EXPOSURE at NDSS [24], we deployed it as a public service that analyzes the passive DNS data obtained from SIE to report malicious domains used in the wild on a daily basis. EXPOSURE has been running since 28 December 2010 and reporting thousands of domains that have been involved in various malicious activities on *http ://exposure.iseclab.org*. In this section, we will present some statistics about the malicious domains detected by exposure during a period of ten months.

During the period of then months, EXPOSURE detected 58412 malicious domains which are mapped to distinct 8562 IP addresses. While at average number of IP addresses associated with a malicious domain is 7, there are several malicious domains that employ hundreds of thousands of IP address. The Figure 5.6 shows the number of IP addresses that are returned in DNS records for the malicious domains identified by EXPOSURE. Note that we have excluded the domains that are resolved to hundreds of thousands of IPs from the graph to be able to clearly show the number of IP addresses mapped to the majority of the malicious domains.

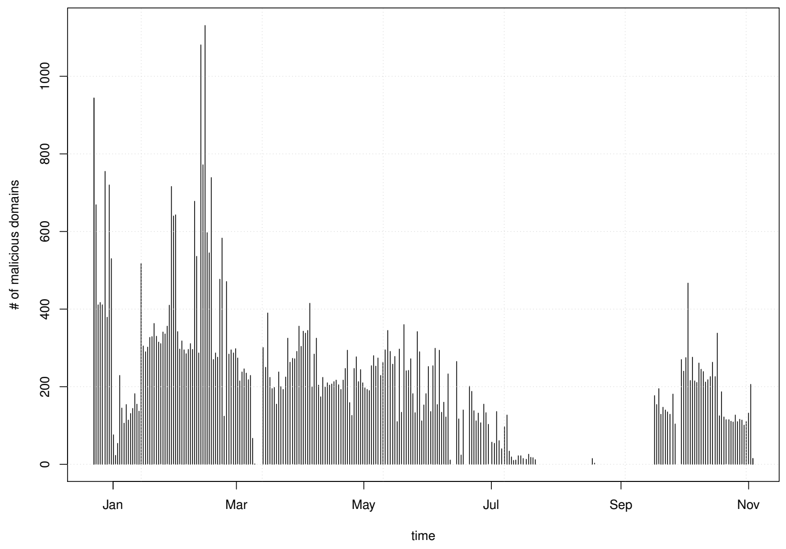


FIGURE *5.7: Number of domains detected on a daily basis by EXPOSURE*

Number of domains detected every day since December, 2010 by EXPOSURE at average is 245. The Figure 5.7 shows the amount of domains detected as malicious over time. As it can be seen, we provided detections on daily basis expect for the period of time between 22 July 2011 and 25 September 2011. The reason of having a two months of gap without detection was the technical problems we faced with to access the DNS data we acquire from SIE.

The top-level domain of the majority of malicious domains detected by EXPOSURE are .INFO, .BIZ AND .ORG as Figure 5.8 shows. Among 58412 malicious domains, 15642 are in .INFO, 14718 are in and 11672 of them are in .BIZ top-level domains. As we stated before, Botnets that emerged in the recent years deploy of DGAs to their bot agents such that the domain of the C&C server to be accessed could be generated automatically at the right time. Typically such domains have shorter lifetime compared to benign domains. 92% of the malicious domains were active less than 30 days. The Figure **??** shows the distribution of the malicious domains with lifetime less than 30 days. The remaining domains are actively used at average more than three months.

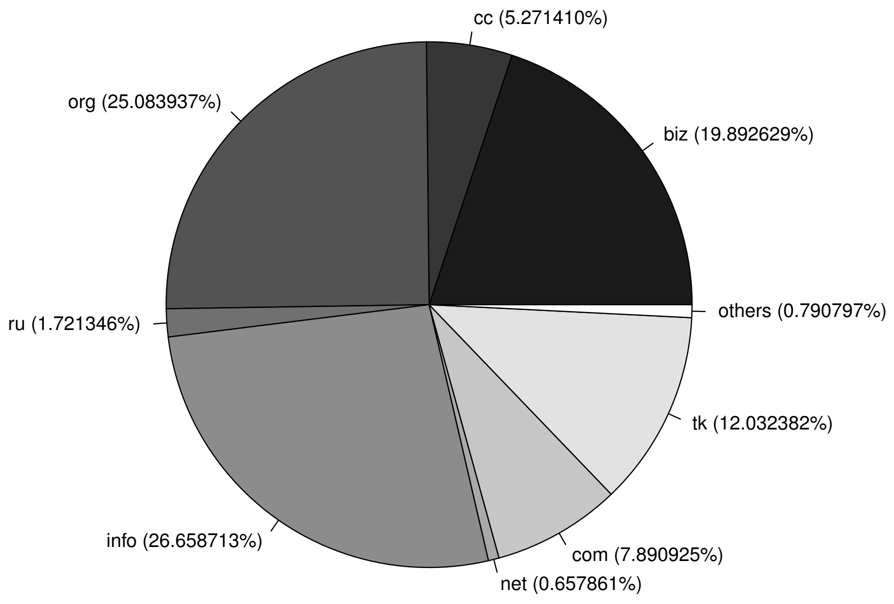


FIGURE *5.8: The percentage of malicious domains with specific top-level domains*

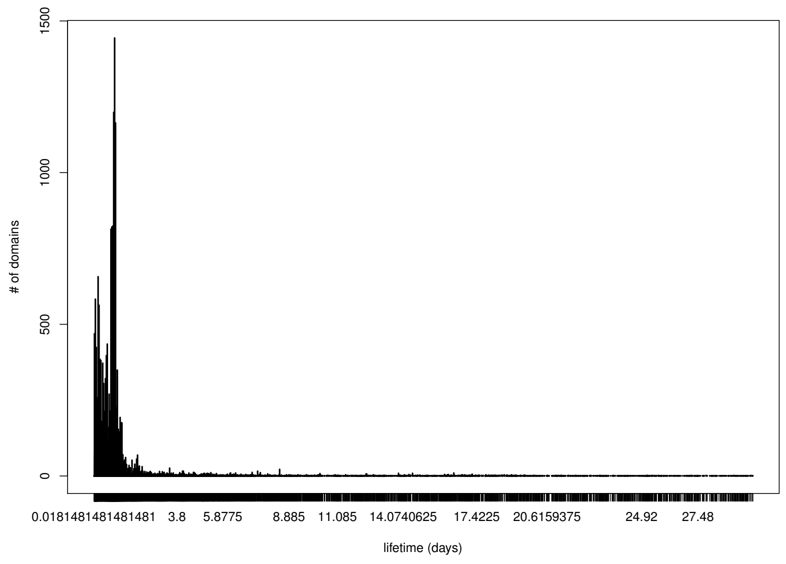


FIGURE *5.9: The distribution of malicious domains according to their lifetime*

5.6 Limitations

A determined attacker who knows how EXPOSURE works and who is informed about the features we are looking for in DNS traffic might try to evade detection. To evade EXPOSURE, the attackers could try to avoid the specific features and behavior that we are looking for in DNS traffic. For example, an attacker could decide to assign uniform TTL values across all compromised machines. However, this would mean that the attackers would not be able to distinguish between more reliable, and less reliable hosts anymore and would take a reliability hit on their malicious infrastructures. As another example, the attackers could try to reduce the number of DNS lookups for a malicious domain so that only a single lookup is performed every hour (i.e., so that the malicious domain is blacklisted). However, this is not trivial to implement, reduces the attack's impact, and requires a high degree of coordination on the attacker's side. Even though it is possible for an attacker to stay below our detection radar by avoiding the use of these features, we believe that this comes with a cost for the attacker. Hence, our systems helps increase the difficulty bar for the attackers, forces them to abandon the use of features that are useful for them in practice, and makes it more complex for them to manage their infrastructures.

Clearly, our detection rate also depends on the training set. We do not train for the family of malicious domains that constitute attacks that are conceptually unknown and have not been encountered before in the wild by malware analyzers, tools, or experts. However, the more malicious domains are fed to the system, the more comprehensive our approach becomes over time.

Note that if the networks that we are monitoring and training our system on are not infected, obviously, we will not see any malicious domains. We believe that we can improve our ability to see more malicious attacks by having access to larger networks and having more installations of EXPOSURE.

Chapitre 6

Botnet Command and Control Server Detection Through Netflow Analysis

Botnets continue to be a significant problem on the Internet. Accordingly, a great deal of research has focused on methods for detecting and mitigating the effects of botnets. Two of the primary factors preventing the development of effective large-scale, wide-area botnet detection systems are seemingly contradictory. On the one hand, technical and administrative restrictions result in a general unavailability of raw network data that would facilitate botnet detection on a large scale. On the other hand, were this data available, real-time processing at that scale would be a formidable challenge. In contrast to raw network data, NetFlow data is widely available. However, NetFlow data imposes several challenges for performing accurate botnet detection.

In this chapter, we present DISCLOSURE, a large-scale, wide-area botnet detection system that incorporates a combination of novel techniques to overcome the challenges imposed by the use of NetFlow data. In particular, we identify several groups of features that allow DISCLOSURE to reliably distinguish C&C channels from benign traffic using NetFlow records (i.e., flow sizes, client access patterns, and temporal behavior). To reduce DISCLOSURE's false positive rate, we incorporate a number of external reputation scores into our system's detection procedure. Finally, we provide an extensive evaluation of DISCLOSURE over two large, real-world networks. Our evaluation demonstrates that DISCLOSURE is able to perform real-time detection of botnet C&C channels over datasets on the order of billions of flows per day.

6.1 System Overview

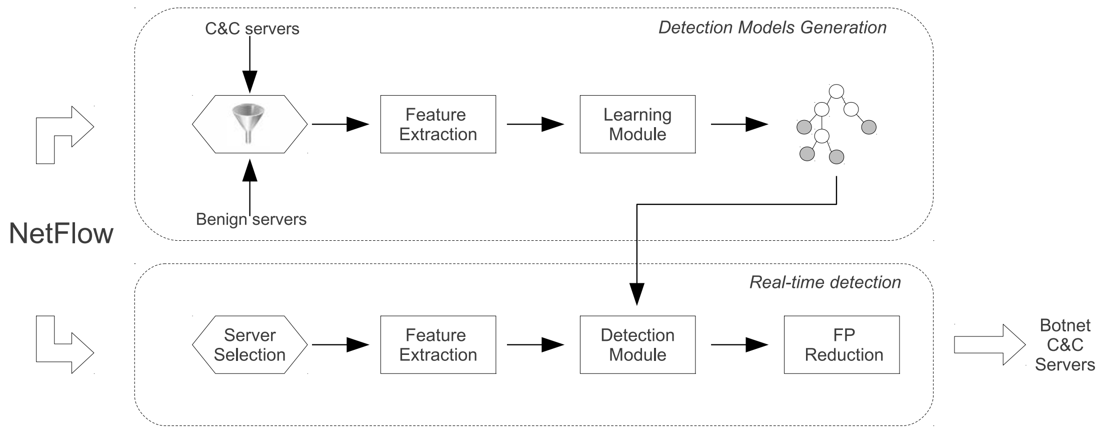


FIGURE *6.1: The system architecture of* DISCLOSURE. *In the training phase (upper half), labeled training samples are used to build detection models. In the detection phase (lower half), the detection models are used to classify IP addresses as benign or associated with C&C communications.*

DISCLOSURE is a botnet detection system designed to identify C&C servers by employing NetFlow analysis. The NetFlow protocol allows for the collection of network flows, or unidirectional sequences of packets exchanged between two peers as part of the same connection. Our work is based on the observation that even though the information that can be extracted from a single NetFlow record is insignificant, it is possible to build statistical models that differentiate the behavior of C&C and benign servers by aggregating large volumes of data collected over an extended period of time.

Figure 6.1 shows an overview of the system architecture of DISCLOSURE. The upper half of the figure describes the detection model generation process, where a supervised machine learning algorithm is used to train models on a subset of NetFlows targeting known benign and C&C servers. Because the training set must be labeled, Disclosure requires as input a list of IP addresses that are known to be either botnet C&C servers or benign servers.

The flows in this labeled data set are first processed by the *feature extraction module*. This module reduces the flows to a number of distinct features, each of which can be classified as belonging to one of three groups : flow size-based features, client access pattern-based features, and temporal features. The specific set of features Disclosure extracts is described in detail in Section 6.2. The features extracted from the training set are then forwarded to Disclosure's *learning module*, which is responsible for building detection models. The learning module can be tuned with several thresholds to obtain an optimal balance between detection and false positive rates.

The bottom half of the graph represents the detection phase, where the models that have been previously generated are applied to unlabeled NetFlows in order to distinguish benign traffic from malicious botnet communication. Since the aim of Disclosure is not to identify bot-infected machines but to detect C&C servers, the first task of the detection phase is to filter those NetFlows that cannot be attributed to a server ; this process is explained in Section 6.4.4. Then, the flows are forwarded to the *feature extraction module*. Finally, the resulting feature vectors are processed by the *detection module* to produce the final list of suspected C&C servers.

Note that the results of Disclosure can be further processed by a *false positive reduction* filter. The goal of this additional module is to correlate the results of Disclosure with the information obtained from other security feeds in order to reduce the probability of misclassification. For example, in Section 6.3, we present a novel technique that associates a reputation score to the autonomous systems to which the C&C servers belong.

6.2 Feature Selection and Classification

In this section, we present the features extracted by Disclosure from NetFlow data in order to detect botnet C&C channels at scale, and discuss why these features are suitable for discriminating between C&C channels and benign traffic. We then describe the particular machine learning techniques we use to build detection models over these features.

6.2.1 NetFlow Attributes

NetFlow is a network protocol proposed and implemented by Cisco Systems [35] for summarizing network traffic as a collection of network flows.

Network elements such as routers and switches capture these NetFlows and forward them to NetFlow collectors.

A network flow is defined to be a unidirectional sequence of packets that share specific network properties (e.g., IP source and destination addresses, and TCP or UDP source and destination ports). Each flow has a number of associated attributes, or summary statistics that characterize various general aspects of its behavior. In this chapter, the NetFlow attributes we analyzed for extracting features to identify command and control servers are : the source IP address, the destination IP address, the TCP source port, the TCP destination port, the start and finish timestamps, and the number of bytes and packets transferred.

Since DISCLOSURE is primarily focused on identifying botnet C&C channels, it is imperative that the system can reliably distinguish servers from clients. Therefore, as an intermediate pre-processing stage, NetFlow data is analyzed by the server classifier that labels each observed IP address according to whether it provides one or more network services. In particular, since multiple services can be made available for each IP address, we represent each server as a set of 2-tuple of IP address and port, $s_i = \langle a, p \rangle$, where $a \in A$ is an IP address, A is the set of all IP addresses, $p \in P$ is a TCP port, and P is the set of all ports.

A common, and legitimate, criticism of early attempts to perform machine learning-based detection over NetFlow data is that the features that were selected were often not *robust* (i.e., in the sense that these features were not necessarily correlated with intrinsic characteristics of how malware manifests in NetFlow data). Hence, the resulting detection systems would often overfit models to the specific behavior of malware represented in the training set — for instance, the particular server port used by a given malware sample. Such features, however, do not generalize to classes of malware such as botnets. For example, using our example of learning a model on server ports, it is clear that the use of a particular server port is not an intrinsic property of botnet behavior. Therefore, the design of DISCLOSURE's feature extractor module emphasizes the selection of those NetFlow attributes that best capture invariants in botnet behavior without resorting to specialization to a particular C&C protocol.

In the following, we describe the specific features DISCLOSURE extracts from NetFlow data.

6.2.2 Disclosure Feature Extraction

Disclosure extracts a number of features from NetFlow data in order to build its detection models. These features can be classified into three broad groups : flow size-based features, client access pattern-based features, and temporal features.

Flow Size-Based Features

The first class of features extracted from NetFlow data are based on *flow sizes*, which simply indicate the total number of bytes transferred in one direction between two endpoints for a particular flow. Our premise for analyzing flow sizes in NetFlow data is that the flow size distributions for C&C servers are significantly and *necessarily* different from flow size distributions for benign servers.

We attribute this difference to several factors. First, the main role of the botnet C&C channel is to establish a connection between the bots and the C&C server in order to transfer commands to the bots, and receive data from the bots. This channel should be both reliable as well as relatively innocuous in appearance. Thus, flows carrying botnet commands or information harvested from infected clients are preferred to be as short as possible in order to minimize their observable impact on the network. Considering that network monitoring tools are widely used and that a botnet's local network impact usually scales linearly with the number of bot infections, tuning for stealth is an important goal. Moreover, due to the limited number of commands in typical C&C protocols, flow sizes tend not to fluctuate significantly. On the other hand, flow sizes generated during accesses to a benign server usually assume a wide range of values.

The preliminary analysis we performed on known sets of benign servers and C&C servers supports our premise. Hence, we designed a set of methods to extract features to detect the behavioral difference between C&C servers and benign servers with respect to flow size.

Disclosure extracts flow size-based features by first grouping all flows according to the server s_i that they originate from or are destined to. Let $s_i \in S$ be a server, and $c_j \in C$ be a client. Then, flow sizes are grouped by time intervals $j = 0, 1, 2, \ldots$, where $F_{i,j}$ denotes a series of flow sizes for flows from endpoint i to j, where endpoints can be drawn from C or S. The time interval is empirically chosen to be 300 seconds. Once this set has been derived, the following feature sets are extracted.

Statistical features. This group of features characterizes the regularity of

flow size behavior over time for both benign and C&C servers. In particular, we extract the mean $\mu^{F_{i,j}}$ and standard deviation $\sigma^{F_{i,j}}$ separately for both incoming and outgoing flows of each server.
Autocorrelation features. Autocorrelation is widely used for cross-correlating a signal with itself in the signal processing domain [27], and is useful for identifying repeating patterns in time series data. A series of flow sizes $F_{i,j}$ can be converted to a time series by ordering sizes by time. Since the autocorrelation function also requires a time series that is sampled periodically as input, we segment the time series by fixed intervals and take the mean over each interval ; empirical testing suggested that a period of 300 seconds is appropriate. Once a periodically sampled time series $\hat{F}_{i,j}$ has been derived from $F_{i,j}$, the series is processed by the autocorrelation function, and features are extracted from the output. Here, we use a discrete autocorrelation coefficient $R_{\hat{F}_{i,j}\hat{F}_{i,j}}$ with lag j normalized by the variance σ^2, where

$$R_{\hat{F}_{i,j}\hat{F}_{i,j}} = \frac{\sum_{i=j}^{n} x_i \overline{x}_{i-j}}{\sigma^2}.$$

The autocorrelation function outputs the correlation results for each period in the time series. This output is further processed by taking the mean and standard deviation over these values to derive the final autocorrelation features.
Unique flow sizes. In addition to the statistical features described above, Disclosure also includes features that count the number of unique flow sizes observed, and performs statistical measurements of occurrence density for each of them during the analysis time. Specifically, an array is constructed in which the elements are the number of occurrences of a specific flow size. Afterward, statistical features are computed over this flow size incidence array to measure its regularity.

Client Access Patterns-Based Features

One typical property of botnets is that the bots frequently establish a connection with the C&C server. These control channels are established for many reasons, including checking whether there is a malicious task to be performed, and sending status messages. These connections tend to be ephemeral, as longer-lived connections might draw undue attention to a bot's presence.

Our basis for selecting features to extract in order to distinguish malicious client access patterns from benign ones is that all of the clients of a C&C server (i.e., bots) should exhibit similar access patterns, whereas the clients of a benign server should not. Since all bots share the same, or nearly identical, malicious program, they tend to access C&C servers similarly unless specifically programmed otherwise. For example, if bots are programmed to query the server every 300 seconds, all of the bots will contact the server with the same frequency. On the other hand, clients of benign services tend to exhibit much more varied patterns due to the vagaries of human action.

Furthermore, even if the client access patterns for a particular server cannot be analyzed because there is little or no data for that server in the data set, differences can be observed between the client access behavior of a single bot and a benign user. However, the strength of this signal in the input data can be attenuated by the NetFlow sampling rate.

Disclosure extracts two sets of features to characterize client access patterns typical of C&C servers and those typical of benign servers.

Regular access patterns. For each server s_i and client c_j, Disclosure prepares a time series $T_{i,j}$ of flows observed during the analysis period. Then, a sequence of flow inter-arrival times $I_{i,j}$ is derived from the time series by taking the difference between consecutive connections ; that is,

$$I_{i,j} = \bigcup_{k=1}^{n} t_{i,j,k} - t_{i,j,k-1},$$

where $t_{i,j,k}$ is the k^{th} element of $T_{i,j}$. Then, statistical features are computed over each inter-arrival sequence, including the minimum, maximum, median, and standard deviation. Finally, we derive the final features for each server s_i by generating statistical features across the set of clients that accessed s_i. This allows Disclosure to not only find regular patterns in clients, but to determine whether the set of clients accessing a server behave similarly.

Unmatched flow density. When a bot is unable to communicate with a legitimate C&C server, it detaches from the rest of the botnet and becomes a zombie. This might happen because the C&C server was shutdown by a botnet detection mechanism, or the IP address of the C&C server has been blacklisted by the administrators of the network in which the bot resides. Since the zombie cannot distinguish between these situations and transient network errors, it continues querying the server. This can result in a significant number of flows to a server that do not have a matching flow in the opposite direction.

It is also possible that a benign server is unreachable for a period of time. However, the behavior of a benign server's clients is significantly different than the behavior of bots that lose access to their C&C servers. This is because when a benign user is aware that a server is offline, it typically

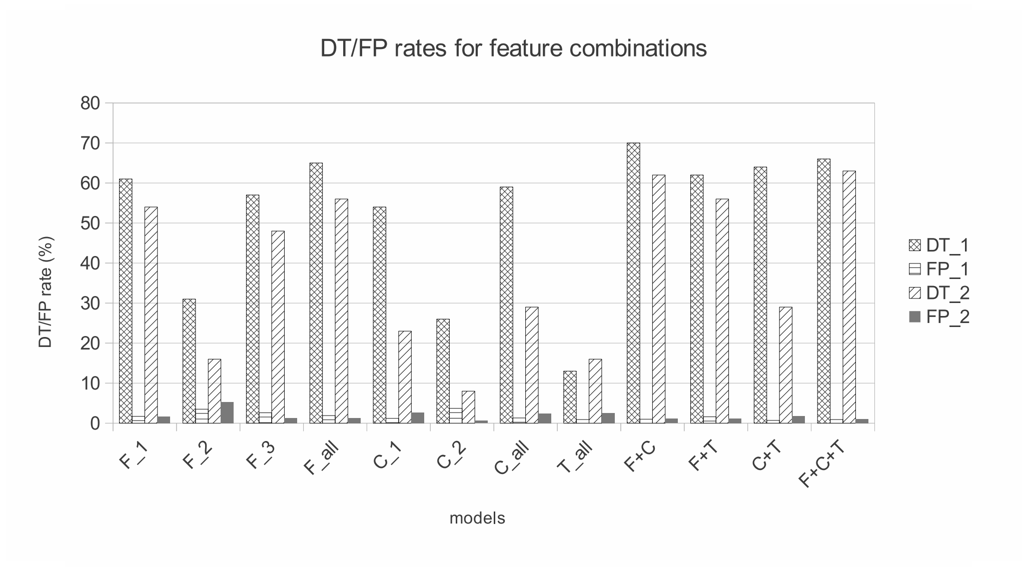


FIGURE *6.2: Detection rates (DT) and false positive (FP) rates for different feature combinations. We note that the DT :FP ratio is most favorable when all features are used in the detection procedure.*

does not insist on continuing to query the server indefinitely. On the other hand, bots tend to exhibit this behavior. Therefore, DISCLOSURE extracts statistics regarding the number of unmatched incoming and outgoing flows to detect this behavior. Specifically, let $U_{i,j}$ be the number of unmatched flows for server s_i in time interval t_j, where

$$U_{i,j} = \sum_{j \in C} \text{abs}\left(|F_{i,j}| - |F_{j,i}|\right).$$

Then, DISCLOSURE derives the mean and standard deviation over a time series of $U_{i,j}$ as a statistical feature.

Temporal Features

Connections to a benign server are subject to diurnal fluctuations representative of the server's user population. On the other hand, connections to C&C servers are dictated by the botmaster, and require no user intervention. As previously mentioned, the majority of botnets configure their bots to contact the C&C server periodically and with relatively short intervals. Therefore, bot-infected machines connect to C&C servers during periods of the day that benign clients do not. For example, many benign servers receive a high volume of traffic during the day, and very little — or nothing — during the night.

To capitalize on this observation, Disclosure extracts a set of temporal features that characterize the variability of client flow volume as a function of time, such that the system can discriminate between uniform client flow distributions indicative of C&C servers and benign traffic that follows well-known diurnal patterns. Specifically, Disclosure segments a time series of client and flow volume by hour-long intervals per server s_i, and calculates statistical features over these.

6.2.3 Building the Detection Models

To build detection models for identifying C&C servers, we experimented with a number of machine learning algorithms, including the J48 decision tree classifier [81], support vector machines [38], and random forest algorithms [64]. Random forest classifiers, known to be one of the most accurate learning algorithms, combine multiple classification methods to achieve more predictive results. In particular, the random forest classifier builds a number of decision trees, where each node in a tree encodes a decision using one or more features that partition the input data. The leaves of each decision tree correspond to the set of possible labels (i.e., $\{\texttt{benign}, \texttt{malicious}\}$), and the output of all of the trees are then ensembled such that the average behavior among all trees is produced as the final decision. In our testing, the best ratio between detection rates (DT) and false positive rates (FP) were produced by the random forest classifier. Furthermore, the classifier is efficient enough to perform online detection in our application. Consequently, Disclosure uses the random forest classifier to build its detection models.

We evaluated our detection models against NetFlow data collected from two networks : a university network (N_1) that does not apply sampling, and a large Tier 1 network (N_2) that samples one out of 10,000 flows. Figure 6.2 shows the detection rates (DT_1 for N_1 and DT_2 for N_2) and false positive rates (FP_1 for N_1 and FP_2 for N_2) for individual features sets, and all possible combinations among different feature sets. The feature sets we evaluated are the set of statistical features extracted from (i) the flow size (F_1) ; (ii) the flow size-based features extracted from the output of the autocorrelation function (F_2) ; (iii) unique flow sizes for each server (F_3) ; (iv) the combination of all flow size-based features (F_{all}) ; (v) the features for characterizing client access patterns (C_1) ; (vi) unmatched flow density (C_2) ; (vii) the combination of all client access pattern-based features (C_{all}) ; (viii) temporal features (T_{all}) ; (ix) the combination of client access pattern and flow size-based features $(F + C)$; (x) the combination of flow size and temporal features $(F + T)$; (xi) the combination of client access pattern and temporal features $(C + T)$; and, finally, (xii) the combination of all feature sets $(F + C + T)$.

Figure 6.2 indicates that individual feature sets are not as effective as combinations of multiple feature sets. Furthermore, increased levels of feature aggregation results in better detection rates with less false positives. Finally, we note that the most promising results were achieved on both data sets by using all possible feature sets as input to the classification process. Hence, Disclosure uses detection models that include all features sets $(F + C + T)$ to detect botnet C&C channels.

6.3 False Positive Reduction

NetFlow data, by its nature, provides limited information about the real activities that are carried out in a network. As a consequence, a botnet detection system based only on the analysis of NetFlow data could produce results that are likely to contain some false positives.

As we explain in the evaluation presented in Section 6.4, Disclosure can be tuned to decrease the overall false positive rate of the decisions to 0.5% or below. However, given the volume of NetFlow data that must be processed every day in large networks, even a misclassification rate less than a fraction of a percent can result in an unacceptably large number of false alarms. Note that some existing malicious activity detection systems have shown to be useful for specific classes of malware or attacks (e.g., fast-flux, domain generation). Clearly, it would be beneficial to correlate the detection results of our system with the results of some previously built systems. Therefore, in our architecture, we include a component that has the aim of correlating the results that Disclosure produces with the public feeds of other malware analysis or detection platforms. The main insight here is that by integrating different data sources, it is possible to further reduce Disclosure's false positive rate to a manageable level.

We have built a reputation-based component for false positive reduction that uses three public services that provide reports about a wide range of malicious activities on the Internet. The first service we make use of is FIRE [10, 94]. FIRE is a system that identifies organizations and ISPs that have been observed to engage in malicious activities. FIRE's website reports detailed information about many autonomous systems (AS), including a maliciousness score, relative rankings among other ASes, as well as the number of C&C servers, exploit servers, and spam and phishing servers the AS has been hosting over time. In our implementation, we separate each type of information into two time series : one representing the current year, and one containing previous historical data. Afterward, we compute statistical features for each time series. For instance, for the time series built from the number of C&C servers observed before 2011, we compute the minimum, mean, and maximum values. After we repeat this step for each time series, we compute a final score by aggregating all the values together by assigning a weight of 0.8 to the value for the current year, and 0.2 to the previous years.

The second public service we use in our false positive reduction component is EXPOSURE [23, 25]. EXPOSURE is a system that uses passive DNS analysis methods to detect malicious domains. EXPOSURE currently analyzes data obtained from a large number of recursive DNS servers, and reports its findings on daily basis. For each domain, it provides the associated IP address list and the autonomous systems in which they are located. Leveraging this information, we count the number of malicious domains detected in each AS and build a reputation score according to the density of maliciousness for each AS reported by EXPOSURE.

The last source of information we use for false positive reduction is Google Safe Browsing [11], a service that reports maliciousness information about a large number of web sites. This tool can also be used to query specific AS numbers to obtain the percentage of web sites in that AS that host malicious services.

For each IP address that DISCLOSURE labels as a potential botnet C&C server, the false positive reduction component fetches the associated AS number and corresponding reputation scores from FIRE, EXPOSURE, and Google Safe Browsing. Each of these individual reputation scores are then aggregated using a weighted linear combination. That is, given the reputation scores r_1, r_2, r_3 and corresponding weights w_1, w_2, w_3 for FIRE, EXPOSURE, and Google Safe Browsing such that $\sum_i w_i = 1$, the final reputation score R is calculated as

$$R = \sum_{i=1}^{3} w_i r_i,$$

where $0 \leq R \leq 1$. If R is below a tunable threshold RepThresh, this indicates that a particular server is located in a network that is historically *not* associated with malicious activities, and the corresponding alert is discarded as a false positive.

We are aware of the fact that the false positive reduction component can introduce an opportunity for the attacker to evade our system. For example, she could place her C&C server in a network with an high reputation score. However, note that this increases the burden on the attacker, and forces her to move away from more vulnerable targets located in ASes with lower reputation scores towards potentially better-protected networks. Therefore, we believe that, on a large scale, this is a favorable result.

6.4 Evaluation

Network	Sampling	Flows per day	Unique IP Addresses
Inter-University Network (N_1)	1 :1	1.2 billion	28 million
Tier 1 ISP (N_2)	1 :10.000	400 million	50 million

TABLE *6.1: Summary statistics for each of the two NetFlow data sets for N_1 and N_2.*

In this section, we present the design and results of several experiments we conducted to evaluate DISCLOSURE's detection accuracy, false positive rate, and performance. We also present deployment considerations, and conclude with a discussion of resilience to evasion.

The accuracy of DISCLOSURE's classification procedure greatly depends upon the environment in which the input NetFlows have been collected. For example, NetFlow collectors placed in a small company network versus those placed in a large ISP will likely observe significantly different volumes of traffic. To bound the storage requirements at each collector, sampling rates might be configured to match the particular traffic volume specific to each site.

To measure how DISCLOSURE responds to varying levels of sampling, we evaluated our system in two distinct environments : a medium-size network connecting multiple universities with no sampling, and a Tier 1 ISP network configured with a sampling rate of 1 :10,000.

In the remainder of this section, we first present the two NetFlow data sets from each of these networks. We then describe the approach we adopted to build our ground truth. Finally, we present and discuss the results of our experiments.

Network	C&C Servers	Benign Servers
University Network (N_1)	892	1489
Tier 1 ISP (N_2)	2000	1742

TABLE *6.2: IP addresses in our labeled data set derived from data observed in N_1 and N_2.*

6.4.1 The NetFlow Data Sets

Our NetFlow data sets were drawn from two separate environments : a university network located in Europe, and an ISP network located in the USA and Japan. Hereinafter, we refer to the university network as N_1 and to the Tier 1 ISP network as N_2.

Table 6.1 shows summary statistics for the two data sets. The N_1 data set was collected for a period of 18 days between the 7th and the 25th of September 2011. The NetFlow data of N_1 is not sampled and, therefore, all network flows present in the monitored network are represented in the data. The sensor in N_1 produced an average of 1.2 billion network flows per day. During this period, we collected 22 billion flows between 28 million unique IP addresses.

In contrast, we collected NetFlow data observed at N_2 for a period of 40 days between the 1st of June 2011 and the 10th of July 2011. The sensors in N_2 were configured to sample flows at a rate of 1 :10,000. The data was harvested by 68 sensors, each of which was responsible for monitoring and forwarding NetFlow traffic collected from specific autonomous systems (ASs). The sensors collected approximately 400 million network flows per day between 50 million unique IP addresses.

6.4.2 The Ground-Truth Data Sets

The accuracy of the classification models generated by a machine learning algorithm greatly depends on the quality of the training set [99]. In our case, to train the features used by DISCLOSURE, we required a ground-truth list containing both known C&C servers and known benign servers.

The malicious server data set consisted of 4295 IP addresses associated with real C&C servers observed in the wild during approximately three weeks preceding our experiments. The list of botnet C&C servers was provided to us by a company that specializes in threats intelligence. This list was manually verified by the company, and potential false positives were eliminated.

We constructed our benign server training set from ranking information provided by Alexa [4]. In this case, we assume that the top 1,000 popular web sites reported by Alexa are not involved in malicious activities and, in particular, are not responsible for hosting botnet C&C servers. Alexa reports the top popular web sites grouped by geographical regions as well. In order to obtain a comprehensive list of benign servers, we combined the "Alexa Top 1000 Global Sites" with the most visited websites in the regions where N_1 and N_2 are located.

Once the benign domains lists were compiled, we resolved each DNS name on both lists to obtain the corresponding list of IP addresses. Note that we executed the DNS queries for each list from the same network geographical locations of the corresponding network (Europe for N_1, and US for N_2). Hence, the number of IP addresses collected for each network is different. This process resulted in 2,958 unique IP addresses for N_1, and 3,047 IP addresses for N_2.

Table 6.2 shows the number of benign and malicious servers in our labeled data set that were observed in the traffic of N_1 and N_2 respectively.

6.4.3 Labeled Data Set Detection and False Positive Rates

In the initial experiment, we evaluated DISCLOSURE's ability to recognize known botnet C&C servers from the ground truth constructed in the previous section. DISCLOSURE's detection rate and false positive rates were measured by generating ROC curves for each data set under two configurations each that controlled the level of input data filtering performed prior to detection.

DISCLOSURE requires a minimum number of observed flows to a particular server in order to provide accurate results. This minimum is a threshold we denote by MinFlows, and can be set by a security administrator according to the volume of traffic at a particular site and any sampling that may be applied. We evaluated two values for MinFlows for each data set : 20 and 50. For each experiment, we excluded any servers that did not have at least one port that received more than MinFlows flows. We then evaluated the accuracy of DISCLOSURE's detection models by performing a 10-fold cross-validation.

We also considered varying the size of the training set as an additional tunable parameter. Figure 6.3 shows a summary of DISCLOSURE's accuracy, measured by computing the area under the ROC curve for different training windows. The curve for N_2 is almost constant. In comparison, the curve for N_1 steadily increases over the first 15 days before plateauing. This is due to

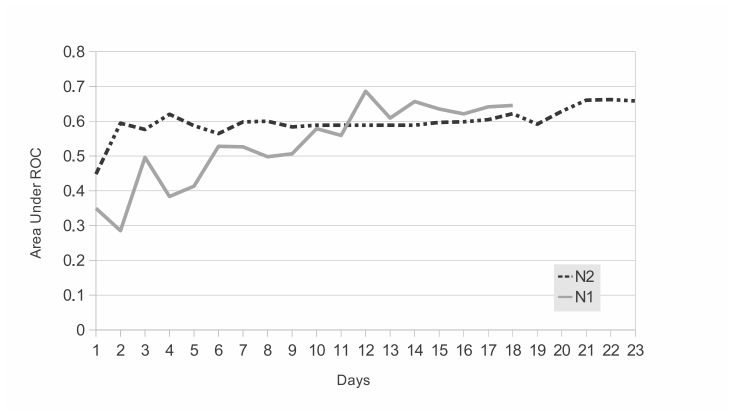
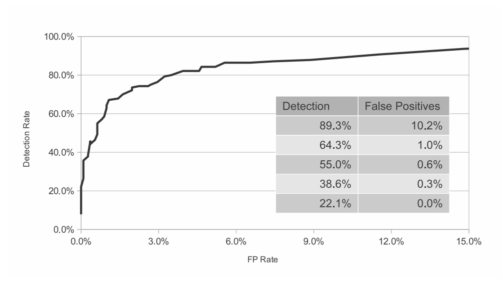


FIGURE *6.3: Area under ROC curves with different training set lengths for N_1 and N_2.*

the fact that the number of known C&C servers observed in the university network is low (see Table 6.2). Therefore, more time is required to collect enough data to properly train the models. For this reason, we decided to train DISCLOSURE with all the available data.

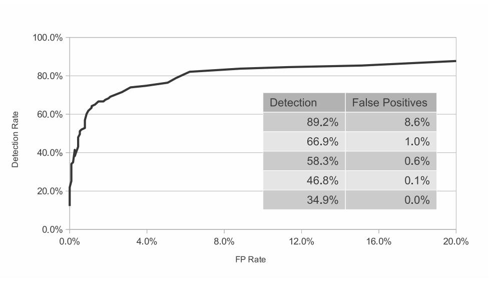
Figure 6.4 shows the individual ROC curves obtained by varying the classification threshold ClassThresh, i.e., the boundary separating benign scores from malicious scores, of DISCLOSURE's detection module. Consequently, each point in the ROC curves represents a possible setup configuration of the system. Security administrators can thus precisely tune DISCLOSURE to achieve a reasonable trade-off between false positives and false negatives based on the traffic characteristics of the network. Each graph also contains a short synopsis of possible working points. For example, configuring the system for a very high detection rate is usually too costly in terms of false positives. On the other end of the scale, it is often possible to achieve a 0% FP rate if we accept the fact that only one out of three C&C servers will be detected.

Despite the differences between the two data sets, the results are similar. For instance, with MinFlows set to 20 flows and ClassThresh tuned to produce a 1% false positive rate, the system detects 64.3% of the C&C servers in the university network and 66.9% in the ISP network. This similarity emerges from the composition of all features, where the individual contribution of each feature is quite different in the two environments. For instance, most of the features are better suited to the unsampled data set, where traffic patterns are clearly preserved. However, some of the features — for instance,



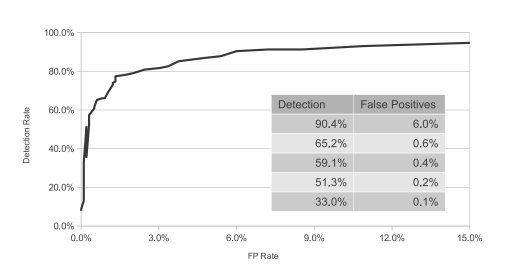
Detection	False Positives
89.3%	10.2%
64.3%	1.0%
55.0%	0.6%
38.6%	0.3%
22.1%	0.0%

(a) N_1 with $\mathsf{MinFlows} = 20$.



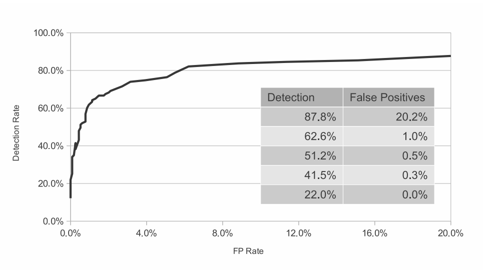
Detection	False Positives
89.2%	8.6%
66.9%	1.0%
58.3%	0.6%
46.8%	0.1%
34.9%	0.0%

(b) N_2 with $\mathsf{MinFlows} = 20$.



Detection	False Positives
90.4%	6.0%
65.2%	0.6%
59.1%	0.4%
51.3%	0.2%
33.0%	0.1%

(c) N_1 with $\mathsf{MinFlows} = 50$.



Detection	False Positives
87.8%	20.2%
62.6%	1.0%
51.2%	0.5%
41.5%	0.3%
22.0%	0.0%

(d) N_2 with $\mathsf{MinFlows} = 50$.

FIGURE *6.4: Classification accuracy for each data set (N_1 and N_2) with $\mathsf{MinFlows} \in \{20, 50\}$.*

the *unmatched flow density* — provide the best results when applied to large networks, even in presence of a high sampling rate. The mixture of these two classes of features makes DISCLOSURE less sensitive to variability in the NetFlow collection environment and, therefore, more robust.

Another important difference between the two experiments is the fact that in the small network (N_1), DISCLOSURE provides better results for a higher value of the minimum flow threshold, while in the large network (N_2) it performs better with a lower threshold. This phenomenon is due to the fact that in the second case, the sensors are only collecting 1 flow out of every 10,000. Therefore, a high value for MinFlows would filter out all small-to-medium size botnets, leaving only a few large ones for the analysis. As a result, the features are now trained on a very few C&C samples and, therefore, tend to produce inaccurate models. This is an important issue to keep in mind when configuring the system. In general, if MinFlows is set too low, the features are exposed to samples that do not show sufficient regularity because an insufficient number of flows are observed in the traffic. If, on the other hand, MinFlows is set too high, the majority of the botnets

Point on the ROC curve	Servers Flagged as C&C (N_1)	Servers Flagged as C&C (N_2)
1.0% FP	12,383	4,937
0.5% FP	7,856	3,166
0.3% FP	6,295	1,958
0.0% FP	132	960

TABLE *6.3: Servers flagged as malicious by* DISCLOSURE *for each of the networks N_1 and N_2 (without incorporating reputation scores).*

are discarded, and the features are trained on too few samples. In both extremes, the result is a set of poorly trained models.

Finally, we manually verified the features of the benign servers that DISCLOSURE wrongly classified as being botnet C&C servers. In several cases, the network or the server were probably malfunctioning, and the clients (in most of the cases less than 10) were repeatedly trying to send the same data over and over again at regular intervals, and receiving no answer back from the server. This behavior, even though not malicious *per se*, is indeed quite similar to that exhibited by bot-infected machines.

6.4.4 Real-Time Detection

In the previous section, we presented the results obtained with labeled data sets containing known benign and botnet C&C servers. In order to apply DISCLOSURE to the remaining unlabeled data, we needed to perform three separate operations.

First, since DISCLOSURE is meant to discover C&C servers and not infected machines, we need to restrict the analysis to the servers only. In order to separate them from the clients, we apply the following heuristic : an IP address belongs to a server if the number of flows directed towards its top two ports (i.e., the two that receives the most connections) account for at least 90% of the flows towards that address. From the count, we removed the ports used less than 3 times to filter out the noise generated by the fact that servers may also have outgoing connections. By adopting this technique, we identified 82,580 servers in N_1 and 530,011 servers in N_2.

The second step consisted of setting the value of the MinFlows threshold. According to the results obtained in the labeled data set, we decided to perform the rest of the experiments with the threshold set to 50 flows for N_1 and to 20 for N_2. After applying the threshold, we were left with 53,426 servers in N_1 and 48,713 in N_2.

Point on the ROC curve	C&C Servers after the Reputation filter (N_1)	C&C Servers after the Reputation filter (N_2)
1.0% FP	1779	1516
0.5% FP	1448	688
0.3% FP	1236	271
0.0% FP	20	91

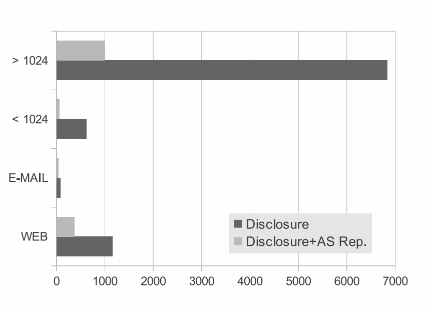
TABLE *6.4: Servers flagged as malicious by* DISCLOSURE *for each of the networks N_1 and N_2 (incorporating reputation scores).*

Finally, we needed to select the operational point on the ROC curve ClassThresh (i.e., the trade-off between detection and false positive rates). Table 6.3 shows the number of servers detected in the two networks obtained with four different configurations of the system.

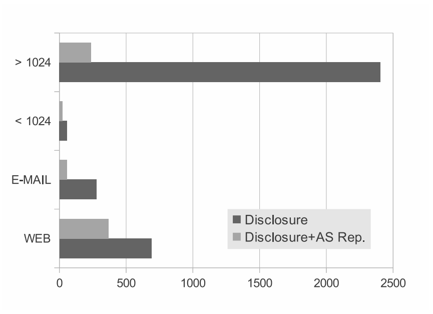
Despite the fact that the various configurations were chosen to minimize the number of false positives generated by the system, the number IP addresses suspected of being C&C servers is still relatively high. Therefore, to further reduce the probability of misclassification, we combined the results of DISCLOSURE with a reputation score based on the information provided by EXPOSURE [23, 25], FIRE [10, 94], and Google Safe Browsing [11]. As explained in Section 6.3, this approach has the effect of narrowing down the results to the servers that have a higher probability of being malicious.

The way in which the reputation score is computed can be tuned according to the desired results and the number of daily alerts that the security administrator can tolerate. The more aggressive the filtering, the smaller the set of IP addresses flagged as C&C servers. In our experiments, we increased the strength of the false-positive reduction until we reduced the amount of alerts to a level that can be manually verified. The results are reported in Table 6.4.

Figure 6.5 shows the ports distributions of the C&C servers detected by DISCLOSURE in the 0.5% false positive setting for N_1 and N_2. The graphs report the two most frequently used protocols : HTTP-related (ports 80, 443, 8080, 8000) and SMTP/IMAP (ports 25, 143, and 993). The remaining ports are grouped in two categories : the reserved ports (0-1023), and the registered and ephemeral ports (1024-65535). This classification is based only on the port number and not on identification of the true protocol. For instance, a botmaster can run a C&C server on port 25 to avoid firewalls, but that does not mean that he will adopt the SMTP protocol as well. It is interesting to note that the majority of the services identified by DISCLOSURE run on



(a) N_1 port distribution.



(b) N_2 port distribution.

FIGURE *6.5: Port distributions of the C&C servers detected by* DISCLOSURE *for both N_1 and N_2, with and without AS reputation scores.*

ports higher than 1024. However, the distribution changes significantly after the false positive reduction is applied. In fact, the reputation system filters out around half of the HTTP services, but cuts between 70 and 90% of the services running on high port numbers.

Finally, we manually investigated the C&C servers detected by DISCLOSURE to gain some insight into the accuracy of the detection models and the reasons for misclassification. To this end, we chose the most conservative configuration : DISCLOSURE configured for 0% FP + Reputation filter. With this setup, during one week of operation, DISCLOSURE reported 91 previously unknown C&C servers on the ISP network, and 20 on the university network.

We first manually queried popular search engines for each of the 111 entries. In 36 cases (32.4%), we found evidence of malware that was related to those IP addresses.[1] The fact that one third of our reports were confirmed by other sources is a strong support of the ability of DISCLOSURE to successfully detect C&C servers. Out of the remaining servers, 30 were associated with HTTP-related ports. After a manual investigation, 7 of them seemed to be legitimate web sites — even though it is unusual that a small real estate company or a personal page in the Philippines would receive the large number of connections we observed in our traffic. 4 pages were default placeholders obtained with a fresh installation of a web server ; the number of NetFlow entries and varying flow sizes is suspicious, although this could be attributed to the web server not having a default virtual host config-

1. This evidence included reports from ThreatExpert, various sandbox malware analysis tools, MaliciousUrl.com, or the offensive IP database.

ured. 4 servers returned errors related to either unauthorized access or bad requests. 3 of the HTTPS servers did not use the SSL/TLS protocol but some other form of binary protocol. The remaining servers were unaccessible at the time we checked them, which was approximately three weeks after the data was collected. Of the non-HTTP services, only 4 were still running at the time the checks were performed. 3 of these appeared legitimate, but the remaining service was a web server located in Russia listening to a non-standard port. Finally, interestingly, 8 servers were located in the Amazon cloud network, which is rapidly increasing in popularity for hosting ephemeral malicious services.

6.4.5 Performance Evaluation

As described in Section 6.1, the detection phase consists of two modules : feature extraction and detection. The detection module is highly efficient, requiring only several minutes to process an entire day's worth of data. Hence, detection performance is constrained by the analysis of input NetFlow data to extract the requisite features for analysis.

However, since the extraction of each feature is an independent process, the feature extraction procedure is an example of an embarrassingly parallel problem that can be easily distributed on multiple machines should the need arise. Nevertheless, even with the large amount of input data for our evaluation networks, we have not found it necessary to parallelize feature extraction. The current prototype implementation of DISCLOSURE consists of a number of Perl and Python scripts, all running on the same server : a 16 core Intel(R) Xeon(R) CPU E5630 @ 2.53 GHz with 24 GB of ram.

In the course of our experiments, we run all individual feature extraction modules in series in 10hours 53minutes for 24 hours of data. Therefore, DISCLOSURE is able to perform at approximately 2x real-time.

6.4.6 Deployment Considerations

To deploy DISCLOSURE to a real network, the administrator should configure three main settings : the minimum flows threshold MinFlows, the classification threshold ClassThresh, and the false positive reduction threshold RepThresh. This setup can be accomplished by performing the following steps :

1. Choose the MinFlows threshold.
   This value should be selected according to the NetFlow sampling rate for the monitored network and the amount of available training data.

If the threshold is set too high, the system will not have enough C&C samples to properly train. But, if it is set too low, the system will train on poor data, and produce inaccurate models.

2. Choose an operational point on the ROC curve for ClassThresh.
This value should be selected according to the traffic volume of the network and the misclassification rate that can be tolerated. On one extreme, the system will be able to detect most of the C&C servers, but it will also generate too many false positives. On the other end of the scale, the system will miss many C&C servers, but the results will be much more precise.

3. (OPTIONAL) Apply and tune the false positive reduction module using RepThresh.
To reduce the number of alerts in large networks, DISCLOSURE can be coupled with other detection or verification techniques. In this chapter, we propose the use of an AS reputation-based score to filter the servers hosted in benign networks. The weights for the constituent reputation systems can be modified to have a more aggressive or a more lightweight filtering component, and the overall reputation score filtering strength can be adjusted by setting RepThresh.

6.4.7 Resilience to Evasion

The detection approach presented in this chapter is predicated on the assumption that existing botnets often exhibit a regular, detectable pattern in their communication with the C&C server. However, we have not discussed how strong this requirement is and how difficult it might be for an attacker to perturb this regularity to avoid detection.

To answer this question, we designed two botnet families (hereinafter B_1 and B_2) that attempt to evade our system by inserting a random delay between consecutive connections and a random length padding in each flow. In our implementation, we employed two different randomization functions. The first randomization function produces uniformly distributed values on a fixed range. This is intended to model a botnet in which the programmer uses a random number generator to select a value from a fixed range. The second family adopts a more sophisticated approach and generates random numbers from a Poisson distribution. In this case, we model a more complex scenario in which the botmaster tries to mimic the flow inter-arrival times of benign services, which are known to be well-approximated as a Poisson process [60].

In our experiment, we generated 300 C&C servers for both B_1 and B_2. First, we randomly specified the size of each botnet and the duration of its activity. Afterward, we created synthetic NetFlow data for each server, using one of the aforementioned randomization functions to generate random flow sizes and intervals between consecutive flows.

Each botnet was created according to the following parameters :

Botnet lifetime	1 - 33 days
Number of bots	1,000 - 100,000
Flow sizes	4 - 3076 bytes
Delay between flows	1 min - 1 hr

The only significant difference between the two botnet families is that for B_1, the delay between consecutive flows between each bot and the the C&C server was a uniformly-distributed random value between 1 minute and 1 hour. For B_2, the delay was, instead, drawn from a Poisson distribution whose *mean* was randomly chosen in the 1 minute to 1 hour range. We decided to set 1 hour as an upper bound since, in order to maintain a reasonable flexibility and control over the botnet, a botmaster must be able to to send commands to the infected machines with a delay that is no longer than an hour or two.

Finally, we added the synthetically-generated NetFlows to our labeled data set and re-ran the classification evaluation using a 10-fold cross-validation. In both cases, Disclosure was able to detect all the experimental C&C servers belonging to B_1 and B_2. In addition, the addition of these synthetic botnets to the training set had the side effect of actually increasing the overall detection rate. In other words, some of the real botnets that were not detected by Disclosure in our normal experiments were detected after we added the synthetic data. This implies that our detection models were not properly trained to detect this kind of variability in the C&C channel behavior. However, by adding many new samples with a randomized behavior to supplement the training set, Disclosure was able to subsequently detect real botnets that present similar access patterns.

Chapitre 7

Concluding Remarks

Malware continues to run rampant across the Internet, and among the myriad forms that modern malware can assume, botnets represent one of the gravest threats to Internet security. Through the large-scale compromise of vulnerable end hosts, botmasters can both violate the confidentiality of sensitive user information — for instance, banking or social network authentication credentials — as well as leverage groups of bots as an underground computational platform for performing other illicit activities.

In this thesis, we propose three network-based botnet detection techniques. Each technique models the detections by analyzing different types of network data : the first detection technique performs packet level inspection. The second one analyzes the DNS traffic to find the domains that are abused for different kinds of malicious purposes including being assigned for the command and control servers. And finally, the last one detects command and control servers by analyzing NetFlow data.

With the first technique, we proposed a system that aims to detect bots, independent of any prior information about the command and control channels or propagation vectors, and without requiring multiple infections for correlation. Our system relies on detection models that target the characteristic behavior of every bot – the fact that it receives commands from the botmaster to which it responds in a specific way. A key feature is that these detection models are generated automatically. To this end, our system observes the network traffic that is generated by actual bot instances in a controlled environment. In these traffic traces, we first identify points in time that likely correspond to response activity. Then, we extract the corresponding commands that trigger these activities. We have implemented the proposed approach and demonstrate that it can extract effective detection models for a variety of different bot families. These models are precise in describing the activity of bots and raise very few false positives.

The second approach (EXPOSURE) that we presented in this thesis employs large-scale, passive DNS analysis techniques to detect domains that are involved in malicious activity. We use 15 features that we extract from the DNS traffic that allow us to characterize different properties of DNS names and the ways that they are queried. Our experiments with a large, real-world data set consisting of 100 billion DNS requests, and a real-life deployment for two weeks in an ISP show that our approach is scalable and that we are able to automatically identify unknown malicious domains that are misused in a variety of malicious activity (such as for botnet command and control, spamming, and phishing).

The last botnet detection method we developed specifically focuses on the detection of the botnet command and control servers. We presented Disclosure, a large-scale, wide-area botnet detection system that incorporates a combination of novel techniques to overcome the challenges imposed by the use of NetFlow data. In particular, we identify several groups of features that allow Disclosure to reliably distinguish C&C channels from benign traffic using NetFlow records (i.e., flow sizes, client access patterns, and temporal behavior). To reduce Disclosure's false positive rate, we incorporate a number of external reputation scores into our system's detection procedure. Finally, we provide an extensive evaluation of Disclosure over two large, real-world networks. Our evaluation demonstrates that Disclosure is able to perform real-time detection of botnet C&C channels over datasets on the order of billions of flows per day.

Appendices

Annexe A

Résumé étendu

L'époque où l'Internet utilisé á un réseau universitaire sans activité malveillante est révolue depuis longtemps. Aujourd'hui, Internet est devenu un infrastructures critiques, et il joue maintenant un rôle crucial dans la communication, la récupération des finances, du commerce et de l'information.Il a été rapporté qu'il ya plus de 2,7 milliards de pages Web sur le Internet maintenant.

Malheureusement, tant que la technologie devient populaire, il attire aussi personnes ayant des intentions malveillantes. En fait, la criminalité numérique est une culture défi pour les organismes d'application de la loi. Comme attaques Internet sont facile á lancer et difficiles á retracer, de tels crimes ne sont pas faciles pour poursuivre et traduire en justice. En conséquence, il ya une grande incitation pour les cyber-criminels á s'engager dans malveillants á but lucratif activité illégale sur l'Internet. Malheureusement, le nombre et la sophistication des attaques Internet n'ont cessé d'augmenter dans les dix dernières années.

Un outil de prédilection des criminels numériques d'aujourd'hui sont bots. Un bot est un type de malware qui est créé avec l'intention de compromettre et prendre le contrôle des hôtes sur Internet. Il est généralement installé sur l'ordinateur de la victime par soit exploitant une vulnérabilité du logiciel dans le navigateur web ou le système d'exploitation ou en utilisant des techniques d'ingénierie sociale pour astuce de la victime dans l'installation du robot elle-même. Comparé á d'autres types de logiciels malveillants, la caractéristique distinctive d'un bot est son capacité á établir un canal de commande et de contrôle qui permet un attaquant de contrôler á distance ou de mettre á jour un compromis machine. Un certain nombre de machines zombies qui sont regroupées sous le contrôle d'une entité unique et malveillants (Appelé le botmaster) sont considérés comme un botnet. Ces botnets sont souvent maltraitées que les plateformes de lancer un déni de service attaques, pour envoyer des spams ou pour héberger arnaque pages. L'utilisation d'un botnet, les attaquants aussi généralement voler des informations sensibles sur la machine d'une victime (par exemple, numéros de carte de crédit, chat, les identifiants du compte sociale du réseau, etc.)

Les botnets ont également été signalés comme ayant été utilisées dans des attaques contre nations - á la fois volontairement, et par coïncidence. Par exemple, dans 2007, il y avait un bot basé délibérée et organisée distribués attaque par déni de service contre des infrastructures critiques de l'Estonie. En 2009, les ordinateurs infectés Conficker des trois armées européennes. Ainsi, avions de chasse franais ont été empêchés de décoller, les réseaux de l'armée britannique et allemand bases ont été partiellement l'arrêt, et un certain nombre de forces de police britannique a d débrancher leurs réseaux de l'Internet. Récemment, le botnet Stuxnet a attaqué une infrastructure critique d'un particulier nation. En fait, Stuxnet semble avoir été écrit spécifiquement pour attaquer une marque particulière de surveillance Contrôle et acquisition de données (SCADA). Les systèmes SCADA sont généralement responsables de l'exploitation des composantes clés de la puissance les plantes, les pipelines, les réseaux de distribution d'électricité, et d'autres similaires les systèmes industriels.

Les moyens traditionnels de défense contre les robots collecteurs de s'appuyer sur l'anti-virus (AV) du logiciel installé sur les machines des utilisateurs finaux ainsi que d'autres types de logiciels malveillants. Malheureusement, comme l'existence de botnets nombreuses démontre, ces systèmes sont insuffisants. La raison est qu'elles s'appuient sur les signatures des échantillons connus, bien documentée limitation qui rend difficile suivre l'évolution rapide des logiciels malveillants. Afin d'atténuer ce limitation, un certain nombre de systèmes de défense basés sur l'hôte ont été introduits. Ces systèmes utilisent statique ou dynamique des techniques d'analyse de code pour capturer le comportement des programmes inconnus. En comparant les observer comportement à un modèle qui spécifie les caractéristiques de certains types de logiciels malveillants, les instances jusque-là inconnues du code malveillant peut être identifiés. Cependant, bien qu'utiles, ces systèmes sont problématiques dans pratique, car ils encourent une surcharge d'exécution considérables et nécessitent chaque utilisateur à installer la plate-forme d'-

analyse.

Afin de compléter les techniques d'analyse basées sur l'hôte, il est souhaitable d'avoir un système de détection basé sur le réseau disponible qui peut surveiller le réseau du trafic pour les indications de machines zombies. En pratique, les systèmes de détection basée sur le réseau sont plus préférable que celles basées sur l'hôte. Une raison est que les systèmes de détection basée sur le réseau avoir une visibilité complète sur le réseau tout en techniques basées sur l'hôte cible uniquement les personnes seules. Malheureusement, dans la sécurité écosystèmes, les utilisateurs sont généralement le maillon faible. Par conséquent, les administrateurs réseau emploient souvent des systèmes de détection basée sur le réseau qui peut être déployée à un point de vue dans le réseau. De cette façon, ils sont capables de surveiller les activités du réseau de tous les individus sans donner privelages aux ordinateurs qui ont déjà été protégée par un basé sur l'hôte système de détection de logiciels malveillants.

Une autre raison est que la nature des botnets est sujette à des réseaux à base systèmes de détection. Le trait caractéristique de botnets est la commande et le protocole de contrôle qu'ils adoptent. Comme le infrustructure commandement et de contrôle repose sur un réseau de communication, donc observable dans le trafic réseau, il est considéré comme l'endroit sensible de botnets.Même si botmasters appliquer certaines techniques de masquage à cacher la sémantique de la commande et le protocole de contrôle, car il est transmis sur le réseau, les paquets de rester dans le réseau.

Pour ces raisons, dans cette thèse, nous proposons trois basées sur le réseau techniques de détection botnet. Chaque technique de modèles les détections par l'analyse différents types de données du réseau : la technique de détection effectue d'abord Packet Inspection niveau. Le second analyse le trafic DNS pour trouver les domaines qui sont maltraités pour les différentes sortes de fins malveillantes, y compris d'être affecté pour les serveurs de commande et de contrôle. Et enfin, le dernier détecte les serveurs de commande et de contrôle en analysant les données NetFlow.

A.1 Détection Botnet Par Packet Inspection Réseau

Des travaux antérieurs á la détecter les robots collecteurs de procéder á l'inspection de paquets au niveau du réseau a progressé le long de deux axes principaux : Le premier axe de recherche utilise la corrélation verticale techniques.Ces techniques se concentrer sur la détection de l'individu bots, habituellement en vérifiant les schémas de trafic ou de contenu qui révèlent commandement et de contrôle du trafic ou malveillant, bots liés activités.Ces systèmes nécessitent une connaissance préalable sur la canaux de commande et de contrôle et les vecteurs de propagation des bots qu'ils peuvent détecter.Par exemple, Rishi analyse le trafic IRC pour les pseudos qui sont fréquemment utilisés par les robots, tandis que le système proposé par Binkley et contrôles Singh pour suspectes les statistiques du trafic IRC.BotHunter est plus avancés dans l'outil combine des alertes á la fois basée sur les anomalies et la signature des systèmes de détection d'intrusion afin d'identifier bot trafic lié.Néanmoins, comme souligné par les auteurs dans un suivi de papier, BotHunter repose sur le fait que le comportement bot "suit un cycle de pré-défini d'infection de vie modèle de dialogue", ce qui est orientée vers les robots collecteurs qui utilisent balayage aléatoire et les commandes bot connu.

Le deuxième axe de recherche pour détecter les robots collecteurs utilise horizontale la corrélation des approches pour analyser le trafic réseau pour les modèles qui indiquent que deux ou plusieurs hôtes se comportent de façon similaire.Ces similaires modèles sont souvent le résultat d'une commande qui est envoyée á plusieurs les membres de la même botnet, causant des bots á réagir dans le même la mode (par exemple, en commençant á numériser ou d'envoyer des mails).Le saillants propriété de techniques qui utilisent la corrélation horizontales, telles que BotSniffer, BotMiner, et TAMD, est qu'ils ne nécessitent pas une priori des informations sur la manière dont le commandement et le contrôle canal est mis en uvre.L'inconvénient de ces approches est qu'elles ne peut pas détecter les robots collecteurs individuels.C'est, il est nécessaire qu'au moins deux hôtes dans les réseaux de surveillance sont membres de la même botnet.Compte tenu de la tendance générale á la plus petite des botnets et la possibilité pour un botmaster d'attribuer deux robots de la gamme d'un même réseau botnets différents, c'est un inconvénient important.

La techniquebotnetpremière détectionprésenté dans cette thèsepropose une approchesimplede détection pour identifier, machineszombiessans aucune connaissance préalablesur le commandement et mécanismes de contrôle oude la manière dontse propageun bot.notredétection modèles'appuie surle comportement caractéristiqued'un bot, quiest qu'il (a)reçoit les commandes dubotmaster,et (b)s'acquitte de certaines actions en réponse áces commandes.Semblable ádes travaux antérieurs, nous supposons quela commandeet les résultatsdes activitésde réponsedans une sorte de réseau de communicationqui peuvent être observées.

L'idée de base de notre système est que nous pouvons générer des modèles de détection en observant le comportement des robots qui sont capturés á l'état sauvage. Plus précisément, en lançant un bot dans un environnement contrôlé et l'enregistrement de son activité sur le réseau (*traces*), nous pouvons observerles commandements que ce bot reçoit ainsi que les réponses correspondantes. á cette Finalement, nous présentons les techniques qui nous permettent d'identifier les pointsdans un réseau de traces qui ont probablement en corrélation avec l'activité de réponse. Ensuite, nous analysons la trafic qui précède cette réponse pour trouver le correspondant de commande. Basé sur les observations de commandes et les réponses, nous générer des modèles de détection qui peuvent être déployés pour analyser le trafic réseau pour une activité similaire, en indiquant le fait qu'une machine est infectée par un bot. Notre approche produit des modèles de détection spécifiques qui sont adaptés aux familles bot ou des groupes de robots liés par une commune C&C infrastructures. Parce que le système est automatisé, cependant, il est facile pour générer rapidement de nouveaux modèles pour les robots qui mettent en oeuvredes commandes nouvelles et réponses. Ceci est indépendant de toute connaissance préalable de la protocole ou les commandes que le bot utilise.

A.2 Détection Botnet par DNS Analyse Passive

Depuis le malware est apparu á l'état sauvage, il ya une continue-course entre les bras auteurs de programmes malveillants et les mécanismes de défense des logiciels malveillants.Chaque fois un système de détection de malware a été développé, malware lui-même modifié pour échapper á ces systèmes.En conséquence, ce bras-course causé le malware évolution.

Les botnets sont un autre type de malware. Ils ont également développé des techniques pour déjouer les mécanismes existants de détection des réseaux de zombies. Dans un premier temps, ils ont appliqué le cryptage ou l'obscurcissement de se cacher le fonctionnement interne de leur C&C infrastructures. Malheureusement, la plupart des systèmes de détection botnet qui effectuent l'inspection des paquets au niveau du réseau, y compris notre système que nous avons décrite ci-dessus, sont limitées dans le fait que ils ne peuvent pas faire face aux botnets masquer/crypter leur trafic. Par conséquent, la nécessité d'un nouveau système de détection complémentaires botnet est évidente.

Un des problèmes techniques que les attaquants face á la conception de leur infrastructures malveillants est la question de savoir comment mettre en uvre un infrastructure de serveur fiable et flexible, et le commandement et le contrôle mécanisme.Ironiquement, les attaquants sont confrontés á la même l'ingénierie globale défis auxquels font face les entreprises qui doivent maintenir une grande infrastructure de service distribué et fiable pour leurs clients.Par exemple, dans le cas des botnets, le attaquants ont besoin pour gérer efficacement les hôtes distants qui peuvent facilement composent de milliers de compromis de l'utilisateur final des machines.évidemment, si l'adresse IP du serveur de commande et de contrôle est codé en dur dans le binaire bot, il existe un point unique de défaillance du réseau de zombies. C'est, du point de vue de l'attaquant, chaque fois que cette adresse est identifié et est descendu, le botnet serait perdu.

Le Domain Name System (DNS) est un système de nommage hiérarchique pour ordinateurs, des services, ou toute autre ressource connectée á Internet. De toute évidence, car elle aide les internautes á trouver des ressources telles que le Web serveurs, hôtes de diffusion, et d'autres services en ligne, le DNS est l'un des composants de base et le plus important de l'Internet. Malheureusement, en plus d'être utilisé pour d'évidentes fins bénignes, les noms de domaine sont également populaires pour une utilisation malveillante. Par exemple, des noms de domaine sont de plus en plus jouer un rôle pour la gestion du botnet commande et de contrôle serveurs, des sites de téléchargement du code malveillant est hébergé, et le phishing pages qui visent á voler des informations sensibles á partir méfiance victimes.

Afin de mieux faire face á la complexité d'un grand, distribuée infrastructures, les réseaux de zombies ont été de plus en plus faire usage de domaine noms. En utilisant le DNS, botmasters d'acquérir la souplesse nécessaire pour changer l'adresse IP adresse des serveurs malveillants qu'ils gèrent. En outre, ils peuvent cacher leurs serveurs critiques derrière les services de proxy (par exemple, en utilisant Fast-Flux) pour que leur serveur malveillant est plus difficiles á identifier et prendre vers le bas.

L'utilisation de nomsde domainedonnecontrôleursbotnetla flexibilité dela migration de leurs serveursavec facilité.Autrement dit, lesinfrastructures deréseaux de zombiesdeviennent plus "tolérants aux pannes"á l'égard desadresses IPoù ilssont hébergés.

Notre idée fondamentale est que les services malveillants(par exempleles botnets) sont souvent commedépendantes des servicescomme les services-DNSbénigne, être capable de identifier les domaines malveillants dès qu'ils apparaissent de façon significative aider á atténuerles menaces Internet-nombreusesqui découlent debotnets. Aussi, notrehypothèse est quequand on regardede gros volumes dedonnées,requêtes DNSpour lesbénigne et les domainesmalveillantsdoit présentersuffisamment de différencesdans le comportement qu'ilspeuvent automatiquementêtre distingués.

Avec notre techniquede détectionbotnetseconde, nous présentonsune approched'analysepassiveDNS complet et un système de détection, EXPOSURE,avec efficacité et efficience détecter desnoms de domaine quisont impliqués dansdes activités malveillantes.Nous utilisons 15fonctions(dont 9sont nouveaux etn'ont pas étéproposéavant) qui nous permettentde caractériserles propriétésdifférentesdesnoms DNS et les façons dontils sont utilisés (i.e.,interrogée).

Notez que les chercheurs ont utilisé DNS avant comme un moyen d'analyser, mesurer et estimer la taille des botnets existants dans le passé. certains solutions ont ensuite tenté d'utiliser le trafic DNS pour détecter malveillants domaines d'un certain type. Cependant, toutes ces approches sont uniquement concentrées sur des classes spécifiques de programmes malveillants (par exemple, seulement malveillants Fast-Flux services). Notre approche, en comparaison, est beaucoup plus générique et n'est pas limitée á certaines catégories d'attaques (e.g., les botnets seulement).

Dans notre approche, basée sur les caractéristiques que nous avons identifiés et une ensemble d'apprentissage qui contient connue domaines bénignes et malicieux, nous former un classificateur de noms DNS. être capable de surveiller passivement en temps réel le trafic DNS nous permet d'identifier les domaines malveillants qui ont pas encore été révélé par pré-compilé des listes noires. En outre, dans Contrairement aux techniques actives de surveillance DNS que la sonde pour les domaines qui sont soupçonnés d'être malveillants, notre analyse est furtive, et nous n'avons pas besoin de déclencher malveillants spécifiques activité en vue d'acquérir des informations sur le domaine. l' l'analyse furtive que nous sommes en mesure d'effectuer a l'avantage que nos adversaires, les cyber-criminels, n'ont aucun moyen de bloquer ou d'entraver l'analyse que nous effectuons (par opposition á des approches comme dans).

Á ce jour, un seul système a été proposé, qui vise á détecter domaines malveillants générique utilisant l'analyse passive DNS. Dans un travaux simultanés et indépendants qui a été très récemment présenté par Antonakakis et al., Les auteurs présentent Notos. Notos assigne dynamiquement les scores de réputation des noms de domaine dont la malveillance n'a pas encore été découverts. En comparaison, notre approche n'est pas dépendante de grandes quantités de malveillance historique données (par exemple, les adresses IP des serveurs infectés antérieurement), nécessite moins de temps de formation, et contrairement á Notos, est également capable de détecter malveillants domaines qui sont mappés á un nouvel espace d'adressage á chaque fois et jamais utilisé pour d'autres fins malveillantes á nouveau.

A.3 Détection des Serveurs de Commande et de Contrôle en Analysant les Données du Flux Réseau

Bien que EXPOSURE se comporte bien dans la détection deréseaux de zombiesqui utilisentdes noms DNS pour contacter le serveur de commande et de contrôle, un problème actuel du système est qu'il est evadable si l'utilisation du réseau de zombies DNS est conçu pour êtresemblable á serveurs bénigne (par exemple, des valeurs normales de TTL, une cartographie á un petit nombre d'adresses IP, etc.). Un deuxième problème avec DNS basé sur la détection est que la demande du client á un domaine DNS malicieux ne signifie pas nécessairement indiquer que le client a été infecté. En fait, une telle demande pourraient avoir été simplement causée par une tentative d'infection ont échoué. Par ailleurs, il ya aussi une chance que le domaine demandé n'est pas vraiment malveillants (par exemple, si les attaquants sont l'aide d'une bénigne, domaine compromis pour héberger un serveur de commande etde contrôle). Une troisième problème est que une requête DNS pour un domaine malveillant pourrait être mis en cache et de par conséquent, pourraient ne pas apparaître dans le trafic surveillé DNS. En conséquence, connexions par les hôtes compromis sur le serveur de commande et de contrôlepeuvent être manquées.

Un autre problème tout basé sur le réseau de détection précédente botnet est que les techniques de parts ils ne s'adaptent pas au-delá d'un seul domaine administratif, tout en conservant la précision de détection utile, même si, ils sont efficaces dans certaines circonstances. Cette limitation restreint l'application de systèmes automatisés de détection d'botnet les entités qui sont informés ou assez motivés pour les déployer. Ainsi, nous ont l'état actuel du botnet d'atténuation, où de petites poches de l'Internet sont assez bien protégés contre l'infection alors que la majorité des critères d'évaluation restent vulnérables.

Cette situation n'est pas idéale. Les botnets sont un problème sur Internet á l'échelle qui s'étend domaines administratifs individuels et, par conséquent, un problème qui nécessite une L'échelle d'Internet solution. En particulier, les botnets peuvent continuer á faire des ravages sur l'Internet, malgré le déploiement de systèmes de détection localisée par en se concentrant sur la propagation á travers moins bien protégé les populations.

Deux des principaux facteurs empêchant le développement de efficaces á grande échelle, étendu de systèmes de détection botnet sont apparemment contradictoires. D'un côté, la main, le résultat technique et administratif des restrictions dans un général indisponibilité du réseau de données brutes qui faciliterait la détection sur un botnet á grande échelle. D'autre part, étaient présent les données disponibles, traitement en temps réel á cette échelle serait un formidable défi. Alors que la source de données idéale pour Détection botnet de grande envergure n'existe pas actuellement, il ya, cependant, une source alternative de données qui est largement disponible aujourd'hui : NetFlow données.

Données NetFlow est souvent capturé par les FAI en utilisant un grand ensemble distribué de collectionneurs d'audit et de suivi de la performance á travers les réseaux backbone. Alors qu'il est contraire extrêmement attractif, les données NetFlow impose plusieurs défis pour effectuer la détection précise botnet. Tout d'abord, et peut-être la plus critique, enregistrements NetFlow ne comprennent pas les charges utiles des paquets, mais plutôt des flux enregistrements sont limités á des métadonnées globales concernant un flux réseau tel que le durée de débit et le nombre d'octets transférés. Deuxièmement, les enregistrements NetFlow sont half-duplex, c'est-ils seulement enregistrer une direction d'une connexion réseau. Troisièmement, les données NetFlow sont souvent collectées par échantillonnage duréseau surveillé, souvent á des taux de plusieurs ordres de grandeur ou plus, retiré de la circulationréelle.

Chacune de ces caractéristiques des données NetFlow complique l'élaboration de un détecteur de botnet effectif sur ce domaine. Le détecteur doit être capable de distinction entre le trafic réseau et malveillants sans accès aux charges du réseau, qui est le composant de données réseau qui transportedirectement preuve de comportements malveillants. Le détecteur doit également être capable de reconnatre signaux faibles indiquant la présence d'un botnet á cause des effets combinés de half-duplex capture et d'échantillonnage agressif.

Comme un roman de suivi du travail á nos précédentes techniques de détection botnet, dans la troisième partie de la thèse, nous présentons une grande échelle, une vaste zone botnet système de détection qui incorpore une combinaison de techniques novatrices pour surmonter les défis imposés par l'utilisation des données NetFlow. En particulier, nous d'identifier plusieurs groupes de caractéristiques qui permettent de façon fiable DISCLOSURE distinguer les canaux de commande et de contrôle du trafic á l'aide bénigne enregistrements NetFlow : (i) débit tailles, (ii) schémas d'accès client, et (iii) le comportement temporelle. Nous démontrer que ces fonctionnalités ne sont pas seulement efficaces dans la détection de courant canaux de commande et de contrôle, mais que ces caractéristiques sont relativement robustes contre les attendus botnets contre l'avenir pourrait déployer contre notre système. Par ailleurs, ces caractéristiques ne sont pas conscients de la structure spécifique de commandement et de contrôle botnet connu protocoles.

Bien que les caractéristiques ci-dessus sont suffisants pour capturer coeur caractéristiques de commandement génériques et de contrôle du trafic, ils génèrent aussi des faux positifs dans isolement. Afin de réduire DISCLOSURE du taux de faux positifs, nous intégrons une nombre de scores de réputation externes dans la procédure de détection de notre système. Ces signaux supplémentaires fonctionnent comme un filtre qui réduit de fausses DISCLOSURE de taux positif á un niveau où le système ne peut réalistement être déployé sur réseaux á grande échelle.

A.4 Contributions

En résumé, cette thèse fait les contributions suivantes :

– Nous présentons trois basées sur le réseau botnet de techniques de détection qui effectuent leur analyse sur les différents types de données : le trafic du réseau complet, le trafic DNS et le trafic NetFlow.
– Nous présentons un système de détection botnet qui effectue une inspection de paquets au niveau du réseau. Ce système est un mécanisme entièrement automatisé qui génère de détection bot modèles en observant le comportement réel des instances de bot dans un environnement contrôlé, sans faire d'hypothèses sur le commandement et le contrôle mécanismes. Ce travail a été publié dans ESORICS 2009.
– Avec l'expérience nous avons effectué avec la première méthode de détection, nous démontrer la faisabilité de notre approche en générant modèles de détection pour les familles bot diverses (y compris ceux contrôlés via IRC et HTTP, ainsi que P2P). Ces modèles sont efficaces dans la détection des bots avec peu de faux positifs.
– Nous présentons une autre technique d'analyse novatrices pour la détection des domaines malveillants qui est basé sur l'analyse la demande passive DNS. Notre technique ne repose pas sur une connaissance préalable sur le type de services du domaine malveillants offre (par exemple, le phishing, Fast-Flux les services, le spamming, les botnets qui utilisent un algorithme de génération de domaine, etc.) Ce chiffre est nettement différent des techniques existantes qui seule cible de Fast-Flux domaines utilisés dans les opérations de botnet. En outre, notre approche nécessite moins de temps de formation, et moins données d'entranement á Notos, et n'a pas certaines de ses limites. Ce travail a été publié dans le NDSS2011.
– Nous décrivons la mise en oeuvre de notre détection en temps réel

système que nous appelons EXPOSURE. Nos résultats expérimentaux montrent que la technique que nous proposons est évolutive, et est capable de distinguer avec précision entre les malveillants et les domaines bénigne avec un faible taux de faux positifs.
– Nous vous présentons DISCLOSURE, une grande échelle, une vaste zone botnet système de détection qui détecte de manière fiable botnet C&C dans les canaux facilement accessibles les données NetFlow utilisant un nouvel ensemble de données statistiques robustes fonctionnalités. En particulier, DISCLOSURE n'assume pas *a priori* notamment la connaissance de C&C protocoles.
– Nous intégrer plusieurs systèmes de réputation externes dans DISCLOSURE de procédure de détection pour affiner la précision du système.
– Nous évaluons DISCLOSURE plus de deux réseaux du monde réel, et démontrer sa capacité á détecter á la fois connus et inconnus botnet C&C serveurs sur des échelles non précédemment atteint.

Bibliographie

[1] RFC 1794 - DNS Support for Load Balancing. `http://tools.ietf.org/html/rfc1794`, 1995.

[2] RFC1834 - Whois and Network Information Lookup Service, Whois++. `http://www.faqs.org/rfcs/rfc1834.html`, 1995.

[3] RFC 1912 - Common DNS Operational and Configuration Errors. `http://www.faqs.org/rfcs/rfc1912.html`, 1996.

[4] Alexa Web Information Company. `http://www.alexa.com/topsites/`, 2009.

[5] DNSBL - Spam Database Lookup. `http://www.dnsbl.info/`, 2010.

[6] Google Safe Browsing. `http://www.google.com/tools/firefox/safebrowsing/`, 2010.

[7] Internet Systems Consortium. `https://sie.isc.org/`, 2010.

[8] McAfee SiteAdvisor. `http://www.siteadvisor.com/`, 2010.

[9] Norton Safe Web. `http://safeweb.norton.com/`, 2010.

[10] FIRE : FInding RoguE Networks. `http://www.maliciousnetworks.org/`, 2011.

[11] Google safe browsing diagnostic page for authonomous systems. `http://www.google.com/safebrowsing/diagnostic?site=AS:as_number`, 2011.

[12] B. Amini. Kraken Botnet Infiltration. `http://dvlabs.tippingpoint.com/blog/2008/04/28/kraken-botnet-infiltration`, 2008.

[13] D. Anderson, C. Fleizach, S. Savage, and G. Voelker. Spamscatter : Characterizing Internet Scam Hosting Infrastructure. In *Usenix Security Symposium*, 2007.

[14] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *19th Usenix Security Symposium*, 2010.

[15] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting malware domains at the upper dns hierarchy. In *20th Usenix Security Symposium*, 2011.

[16] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling. The Nepenthes Platform : An Efficient Approach to Collect Malware. In *9th Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 165–184, 2006.

[17] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *Recent Advances in Intrusion Detection*, 2007.

[18] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes - Theory and Application.* Prentice-Hall, 1993.

[19] U. Bayer. Anubis : Analyzing Unknown Binaries. `http://analysis.seclab.tuwien.ac.at/`.

[20] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *16th Symposium on Network and Distributed System Security (NDSS)*, 2009.

[21] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze : A Tool for Analyzing Malware. In *15th EICAR Conference, Hamburg, Germany*, 2006.

[22] P. Berkhin. Survey of clustering data mining techniques. Technical report, 2002.

[23] L. Bilge. EXPOSURE : Exposing Malicious Domains. `http://exposure.iseclab.org/`, 2011.

[24] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE : Detecting Malicious Domains Using Passive DNS Analysis, 2011.

[25] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure : Finding malicious domains using passive dns analysis. In *18th Annual Network and Distributed System Security Symposium (NDSS'11)*, 2011.

[26] J. Binkley and S. Singh. An Algorithm for Anomaly-based Botnet Detection. In *Usenix Steps to Reduce Unwanted Traffic on the Internet (SRUTI)*, 2006.

[27] G. E. P. Box, G. M. Jenkins, and G. Reinsel. Time Series Analysis : Forecasting and Control. In *3rd eddition Upper Saddle River, NJ : PrenticeHall*, 1994.

[28] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. In *Pattern Recognition*, volume 30, pages 1145–1159, 1997.

[29] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K.Salamatian. Anomaly extraction in backbone networks using association rules. In *ACM Internet Measurement Conference (IMC'09)*, 2009.

[30] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, 2006.

[31] H. Choi, H. Lee, and H. Kim. Botnet detection by monitoring group activities in DNS Traffic. In *7th IEEE International Conference on Computer and Information Technologies*, 2007.

[32] M. Christodorescu and S. Jha. Testing Malware Detectors. In *ACM International Symposium on Software Testing and Analysis (ISSTA)*, 2004.

[33] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-Aware Malware Detection. In *IEEE Symposium on Security and Privacy (Oakland)*, 2005.

[34] S. Chu, E. Keogh, D. Hart, M. Pazzani, and Michael. Iterative deepening dynamic time warping for time series. In *In Proc 2 nd SIAM International Conference on Data Mining*, 2002.

[35] B. Claise. Cisco systems netflow services export version 9, 2004.

[36] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup : Understanding, Detecting, and Disrupting Botnets. In *1st Workshop on Steps to Reducing Unwanted Traffic on the Internet*, pages 39–44, 2005.

[37] M. Cova. Wepawet. `http://wepawet.iseclab.org/`.

[38] N. Cristianini and J. Shawe-Taylor. An introduction to support vector machines and other kernel-based learning methods. In *Cambridge University Press*, 2000.

[39] Cyber-TA. SRI Honeynet and BotHunter Malware Analysis Automatic Summary Analysis Table. `http://www.cyber-ta.org/releases/malware-analysis/public/`, 2007.

[40] D. Dagon, G. Gu, C. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.

[41] J. Davis. Hackers Take Down the Most Wired Country in Europe. `http://www.wired.com/politics/security/magazine/15-09/ff_estonia`, 2007.

[42] M. de Hoon, S. Imoto, J. Nolan, and S. Miyano. Open Source Clustering Software. *Bioinformatics*, 20(9), 2004.

[43] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. In *Proceedings of the 2007 workshop on Large scale attack defense (LSAD 07)*, 2007.

[44] D. Dietrich. Distributed Denial of Service(DDoS) Attacks/tools. `http://staff.washington.edu/dittrich/misc/ddos/`, 2005.

[45] M. Domains. Malware Domain Block List. `http://www.malwaredomains.com/`, 2009.

[46] M. Felegyhazi, C. Kreibich, and V. Paxson. On the potential of proactive domain blacklisting. In *Proceedings of the Third USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, San Jose, CA, USA, April 2010.

[47] J. Francois, S. Wang, R. State, and T. Engel. Bottrack : Tracking botnets using netflow and pagerank. In *IFIP Networking 2011*, 2011.

[48] F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking : Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *10th European Symposium On Research In Computer Security*, 2005.

[49] J. Goebel and T. Holz. Rishi : Identify bot contaminated hosts by IRC nickname evaluation. In *Workshop on Hot Topics in Understanding Botnets*, 2007.

[50] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. H. Kang, and D. Dagon. Peer-to-Peer Botnets : Overview and Case Study. In *1st Workshop on Hot Topics in Understanding Botnets*, April 2007.

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

[52] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter : Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *16th Usenix Security Symposium*, 2007.

[53] G. Gu, J. Zhang, and W. Lee. BotSniffer : Detecting Botnet Command and Control Channels in Network Traffic. In *15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[54] J. John, A. Moshchuk, S. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *6th Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[55] C. Kalt. Internet relay chat : Architecture. RFC2810.

[56] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale Botnet Detection and Characterization. In *Usenix Workshop on Hot Topics in Understanding Botnets*, 2007.

[57] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM SIGMOD Conference on Management of Data*, pages 151–162, 2001.

[58] H. Kim and B. Karp. Autograph : Toward Automated, Distributed Worm Signature Detection. In *13th USENIX Security Symposium*, pages 271–286, August 2004.

[59] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based Spyware Detection. In *15th Usenix Security Symposium*, 2006.

[60] D. E. Knuth. Seminumerical algorithms. In *The Art of Computer Programming, Volume 2, Addison Wesley*, 1969.

[61] M. Konte, N. Feamster, and J. Jung. Dynamics of online scam hosting infrastructure. In *In Passive and Active Measurement Conference*, 2009.

[62] C. Kruegel, W. Robertson, and G. Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2004.

[63] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez. Hamsa : Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In *IEEE Symposium on Security and Privacy*, 2006.

[64] A. Liaw and M. Wiener. Classification and regression by randomforest. In *R News*, volume 2/3, page 18, 2002.

[65] M. D. List. Malware Domains List. `http://www.malwaredomainlist.com/mdl.php`, 2009.

[66] Z. B. List. Zeus domain blocklist. `https://zeustracker.abuse.ch/blocklist.php?download=domainblocklist`, 2009.

[67] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. Using machine learning techniques to identify botnet traffic. In *the 2nd IEEE LCN Workshop on Network Security (WoNS'2006)*, 2006.

[68] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists : Learning to detect malicious web sites from suspicious urls. In *Proceedingsof theSIGKDD Conference. Paris,France*, 2009.

[69] M. Mahoney and P. Chan. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[70] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is sampled data sufficient for anomaly detection ? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, 2006.

[71] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Usenix Security Symposium*, 2001.

[72] A. Moser, C. Kruegel, and E. Kirda. Limits of Static Analysis for Malware Detection . In *23rd Annual Computer Security Applications Conference (ACSAC)*, 2007.

[73] J. Nazario and T. Holz. As the net churns : Fast-flux botnet observations. In *International Conference on Malicious and Unwanted Software*, 2008.

[74] C. News. Stuxnet : Fact vs. theory. `http://news.cnet.com/8301-27080_3-20018530-245.html?tag=topStories1`, 2010.

[75] J. Newsome, B. Karp, and D. Song. Polygraph : Automatically Generating Signatures for Polymorphic Worms. In *IEEE Symposium on Security and Privacy*, pages 226–241, 2005.

[76] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. Fluxor : Detecting and monitoring fast-flux service networks. In *Detection of Intrusions and Malware, and Vunerability Assessment*, 2008.

[77] V. Paxson. Bro : A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31, 1999.

[78] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *25th Annual Computer Security Applications Conference (ACSAC)*, 2009.

[79] Phishtank. Phishtank. `http://www.phishtank.com/`, 2009.

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

[81] J. Quinlan. C4.5 : Programs for machine learning. In *Morgan Kaufmann Publishers*, 1993.

[82] J. Quinlan. Learning with continuous classes. *Proceedings of the 5th Australian joint Conference on Artificial Intelligence*, Singapore : World Scientific :343 – 348, 1995.

[83] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Internet Measurement Conference (IMC)*, 2006.

[84] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[85] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, P. Celeda, and V. Krmicek. Improving anomaly detection error rate by collective trust modeling. In *Recent Advances in Intrusion Detection 2008 11th International Symposium*, 2008.

[86] M. Reiter and T. Yen. Traffic aggregation for malware detection. In *DIMVA*, 2008.

[87] K. Rieck, T. Holz, C. Willems, P. Duessel, and P. Laskov. Learning and Classification of Malware Behavior. In *DIMVA*, 2008.

[88] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *13th Systems Administration Conference (LISA)*, 1999.

[89] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *OSDI*, 2004.

[90] A. Sperotto, R. Sadre, and A. Pras. Anomaly characterization in flow-based traffic time series. In *Proceedings of the 8th IEEE international workshop on IP Operations and Management*, IPOM '08, pages 15–27, 2008.

[91] E. Stinson and J. Mitchell. Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods. In *Usenix Workshop on Offensive Technologies (WOOT)*, 2008.

[92] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, C. Kruegel, G. Vigna, and R. Kemmerer. My Botnet is Your Botnet : Analysis of a Botnet Takeover. In *16th ACM Conference on Computer and Communications Security, Chicago, IL*, 2009 November.

[93] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet : Analysis of a botnet takeover. In *ACM Conference on Computer and Communication Security (CCS)*, 2009.

[94] B. Stone-Gross, C. Kruegel, K. Almeroth, A. Moser, and E. Kirda. Fire : Finding rogue networks. In *2009 Annual Computer Security Applications Conference (ACSAC'09)*, 2009.

[95] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting Botnets with Tight Command and Control. In *31st IEEE Conference on Local Computer Networks (LCN)*, 2006.

[96] Symantec. Exploring Stuxnet's PLC Infection Process. `http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process`, 2010.

[97] Symantec. Symantec Threat Report. `http://www.symantec.com/business/theme.jsp?themeid=threatreport`, 2010.

[98] T. Telegraph. French fighter planes grounded by computer virus. `http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html`, 2010.

[99] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, 2009.

[100] T.Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[101] D. Turaga, M. Vlachos, and O. Verscheure. On K-Means Cluster Preservation using Quantization Schemes. In *IEEE International Conference on Data Mining, ICDM09*, 2009.

[102] R. Villamarn-Salomn and J. C. Brustoloni. Bayesian bot detection based on DNS traffic similarity. In *SAC'09 : ACM symposium on Applied Computing*, 2009.

[103] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast ip networks. In *SIG SIDAR Graduierten-Workshop uber Reaktive Sicherheit (SPRING'06)*, 2006.

[104] H. Wang, D. Zhang, and K. G. Shin. Change-Point Monitoring for Detection of DoS Attacks. *IEEE Transactions on Dependable and Secure Computing*, 1(4), December 2004.

[105] F. Weimer. Passive DNS Replication. In *FIRST Conference on Computer Security Incident*, 2005.

[106] I. Witten and E. Frank. *Data Mining : Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005.

[107] J. Wolf. Technical details of Srizbis domain generation algorithm. `http://tinyurl.com/6mdasc`, 2008.

[108] WorldWideWebSize. Size of the WWW. `http://www.worldwidewebsize.com/`, 2010.

[109] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically Generating Models for Botnet Detection. In *14th European Symposium on Research in Computer Security(ESORICS 2009)*, 2009.

[110] G. Yan, Z. Xiao, and S. Eidenbenz. Catching instant messaging worms with change-point detection techniques. In *1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[111] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama : Capturing system-wide information flow for malware detection and analysis. In *ACM Conference on Computer and Communication Security (CCS)*, 2007.

[112] B. Zdrnja, N. Brownlee, and D. Wessels. Passive Monitoring of DNS anomalies. In *DIMVA*, 2007.

[113] H. Zitouni, S. Sevil, D. Ozkan, and P. Duygulu. Re-ranking of Image Search Results using a Graph Algorithm. In *9th International Conference on Pattern Recognition*, 2008.