University of Minho
School of Engineering

Luís Miguel Ferreira Costa Mendonça

Botnet Detection
A Numerical and Heuristic Analysis

January 2012

University of Minho
School of Engineering

Luís Miguel Ferreira Costa Mendonça

Botnet Detection
A Numerical and Heuristic Analysis

Master Thesis
Master in Communication Networks and Services Engineering

Research oriented by:
Henrique Manuel Dinis dos Santos

January 2012

“Information is not knowledge”

Albert Einstein

Table of Contents

List of Figures

List of Tables

List of Equations

Acknowledgements

This research has only been possible thanks to the cooperation and help of some people and institutions.

First of all I have to thank the University of Minho and the Headship of Technologies and Information Systems that allowed me to take this master degree. Secondly I have to thank the Communications Services of University of Minho responsible for giving me access to all the real traffic used in this research.

For the guidance, help, and introduction to the research process, I'd like to thank Henrique Santos that was always there when I needed allowing me to really feel the research way of life and participate in the scientific community. Henrique's words were, and still are, a powerful motivational engine.

Last, but not least, for the backup in all the late-night research shifts, a big thanks to my wife Helena. Not forgetting the constantly growing joy that kept me going, a heart-full thanks to my, already, five year old son Miguel.

1. Introduction

"A journey of a thousand miles begins with a single step." Zen Proverb

Internet security has been targeted in innumerous ways throughout the ages and Internet cyber criminality has been changing its ways since the old days where attacks were greatly motivated by recognition and glory. A new era of cyber criminals are on the move. Real armies of robots (bots) swarm the internet perpetrating precise, objective and coordinated attacks on individuals and organizations. Many of these bots are now coordinated by real cybercrime organizations in an almost open-source driven development resulting in the fast proliferation of many bot variants with refined capabilities and increased detection complexity.

One example of such open-source development could be found during the year 2011 in the Russian criminal underground. The release of the Zeus botnet framework source-code led to the development of, at least, a new and improved botnet framework: Ice IX. [1]

Concerning attack tools, the combination of many well-known techniques has been making botnets an untraceable, effective, dynamic and powerful mean to perpetrate all kinds of malicious activities such as Distributed Denial of Service (DDoS) attacks, espionage, email spam, malware spreading, data theft, click and identity frauds, among others [2].

Economical and reputation damages are difficult to quantify but the scale is widening. It's up to one's own imagination to figure out how much was lost in April of 2007 when Estonia suffered a well-known distributed attack on its internet country-wide infrastructure.

Among the techniques available to mitigate the botnet threat, detection plays an important role. Despite recent year's evolution in botnet detection technology, a definitive solution is far from being found. New constantly appearing bot and worm developments in areas such as host infection, deployment, maintenance, control and dissimulation of bots are permanently changing the detection vectors thought and developed.

In that way, research and implementation of anomaly-based botnet detection systems are fundamental to pinpoint and track all the continuously changing polymorphic botnets variants, which are impossible to identify by simple signature-based systems.

1.1 Goals

The detection of bots (sometimes called zombies or drones when referring to the machines infected) and botnets is critical, and can be made using three distinct methodologies according to [3]: cooperative behavior analysis, attack behavior analysis and signature analysis. This thesis will focus on the first two approaches having in mind that botnet detection should be time-efficient [4] and that the communications between bots and herders can be included in the cooperative behavior analysis group.

Objectively, there is a desire to contribute to all the research already done by the scientific community in the botnet detection and tracking area focusing on network behavior analysis. The advantage of network behavior analysis approach is its relative simplicity and possible operation in a network core with no packet and no host-based inspections.

In the end, this thesis delivers:

1. A deep survey on the existing botnet and network anomalies detection models and approaches focusing in behavior analysis methodology
2. Numerical and heuristic models capable of characterizing network botnet related anomalous activity
3. Prototype of an anomaly and behavior based botnet detection framework capable of identifying botnet activity ideally in real-time scenarios

For that matter, different types of botnets (IRC, HTTP and P2P, among others) are initially studied and followed by the analysis of some existing detection techniques and tools like Honeynet, BotSniffer and BotMinner. Some numerical and heuristic aspects of both normal and bot traffic are then investigated in order to propose a set of traffic parameters aiming fast and precise botnet detection, with, ideally, low false positive rate.

In the end, a botnet detection framework prototype is presented together with usage analysis of the proposed traffic parameters. The detection framework is finally put under test and evaluated under real network traffic.

All the findings in this thesis will, hopefully, be valuable assets to organizations wishing to increase its own network security levels or even enterprises with products and services in the Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) areas.

1.2 Research Methodology

In order to accomplish this investigation, a Design Science approach was undertaken focusing on the seven guidelines provided by Hevner et al in [5]: artifact, problem relevance, research rigor, design as a search process, design evaluation, research contributions, and research communication.

1.2.1 Artifact

This thesis presents a functional botnet detection framework and model. This framework is customizable through the definition of parameters and detection thresholds in order to allow its application in the most different supervised networks scenarios.

1.2.2 Problem Relevance

The botnet phenomenon is a recent, fast growing menace to internet safety so the development of new and improved detection vectors is crucial to internet effective supervision.

1.2.3 Research Rigor

The proposed methods and frameworks were tested using real traffic datasets in order to correctly evaluate its utility and efficacy taking in account data privacy. To verify the proposed framework and model accuracy, detection rates were evaluated.

All results, methods and developed software were also verified several times before any conclusions were taken.

1.2.4 Design as a Search Process and Evaluation, Research Contributions and Communication

An initial part of this work was already submitted to the 10th European Conference on Information Warfare and Security thus pretending to be an academic contribution to this field of study. All the detection framework prototype code produced in this thesis will be submitted to open-source repositories in order to contribute for further research in the matter.

1.3 Document Organization and Notes of Attention

Beyond this introduction where thesis goals, methodology and calendar are presented, the reminder of this paper is organized as follows: in section 2, botnet characterization is done in order to better understand what kind of threats botnets present nowadays. Sections 3 and 4 presents, analyzes and discusses existing botnet detection research vectors as well as some of the related work on the matter.

Section 5 proposes a botnet anomaly-based detection model, presenting at the same time all developed heuristics as well as an implementation prototype for the detection system proposed. The initial research and results of the paper “Botnet Detection: A Numerical and Heuristic Analysis” published in the proceedings of the 10th European Conference on Information Warfare and Security (ECIW’11) begin this section since they were this thesis’ jumpstart and are useful for a better understanding of the thesis research path.

In section 6, the test bed built, corresponding framework testing and results are analyzed in order to validate the detection heuristics considered.

This thesis ends in the 7th section of this document where an analysis and discussion of the proposed detection model is promoted as well as several possible solutions for the weaknesses found.

A very important aspect is that all decimal data presented in this section uses the Portuguese decimal notation with comma (“,”) as decimal separator.

1.4 Research Calendar

The research calendar is shown below.

Tasks	**Approx. Dates**	**Milestones**
Preliminary study of different methodologies used for Botnet detection and related work survey.	Nov/Dec 2010	*
Definition of a botnet characterization model	Jan 2011	
Collection of necessary network traffic datasets for analysis	Jan/Feb	**
Tests on the model and datasets collected. Extraction of numerical properties of the traffic analyzed	Feb/May	
Creation of a set of heuristics and statistics capable of identifying botnet network traffic activity	May/Jun	***
Detection System Prototype based on the defined models and heuristics as well as final tests	May/Oct	
Final thesis writing, revisions and rectifications	Oct/Nov	****
Delivery of full thesis	Dec 2011	*****

*	Thesis draft delivered to thesis coordinator
**	All traffic datasets necessary collected
***	Thesis draft with all the heuristics that will be analyzed in the thesis tested and described as well as a complete related work survey
****	Full thesis draft delivered to thesis coordinator
*****	Final version of the thesis delivered to thesis coordinator

2. Botnet Characterization

“The general who wins the battle makes many calculations in his temple before the battle is fought. The general who loses makes but few calculations beforehand.” Sun Tzu. The Art of War.

Global bot development status and *modus operandi* are analyzed in this section in order to achieve Botnet characterization. This step is fundamental to create coherent and comprehensive botnet detection methods.

Though it is true that botnets exist in a great variety of forms, some types of activity are shared and can be observed in all of them. These stereotyped activities must exist in botnet regular operations in order to make it useful to herders.

This section also presents some basic botnet information besides fundamental vocabulary to understand the rest of this document.

2.1 Definition

For this thesis purpose, botnet is considered to be a group of two or more distinct hosts performing malicious coordinated activities. Coordinated operations are achieved with the help of computer programs running in each host (bots).

2.2 Propagation and Recruitment

By this thesis botnet definition, botnets are built upon recruited hosts and, as in army forces, recruitment can be volunteer and non-volunteer.

Volunteer bot recruitment is not the norm but can be found within some social activists actions. One recent example was the Anonymous and Wikileaks matter where a botnet was built, and grown, upon many of the world citizens' will to participate in the cause of Wikileaks through the "infections" of their own personal computers [6].

Non-volunteer recruitment involves the unawareness of host's owners and normally includes the installation of some form of malware (backdoor, virus or worm among others).

Volunteer recruitment is a basic and straightforward process that simply involves the willingly installation of some binary by the host user. The installed code will put the software and hardware on which is installed under the herder's orders. Non-volunteer recruitment is a far more complex procedure and will be discussed throughout the rest of this subsection.

There are several ways a herder can use to recruit new hosts to its botnet without the host's user consent. They all end up involving some form of user or host vulnerabilities exploitation. Users are vulnerable in the way that they can be tricked into installing malicious programs unconsciously. On the other hand, hosts are vulnerable in the way that they run (sometimes) faulty, misconfigured or insecure applications.

Either way, the host's takeover process is achieved through a computer program executed directly by the host's user or by the herder through a host's vulnerability. This initial action, known as the *Initial Infection* is the kick start for other applications installation such as Virus, Worms, Trojans, Backdoors or Bots. [7]

2.2.1 Initial infection

Host vulnerabilities exploitation can have its origins in human or bot actions. On the human related actions, hacker's attacks on servers' services are in small number when compared to all the user-activated malicious payloads contributing to successful *Initial Infections*. User-activated infections are based in the execution of code accessed through driven-by-downloads, malicious email attachments or infected removable media. Bots, on the other hand, are also known to exploit unsecure or misconfigured services and applications as well as sending SPAM emails containing additional malicious payloads thus contributing to more and more drive-by-download infections.

As mentioned before, misconfigurations and bugs in host services and applications such as browsers are some of the vulnerabilities exploited to run malware droppers[1] on a given target host. Some of these infection vectors are briefly described below.

Exploitation through removable media is one of such infection vectors. This form of infection is achieved when malware present in removable drives is executed by the host's user or by the auto-run feature of some operating systems.

Another widespread infection vector is known as drive-by-downloads. Through social engineering, emails can trick a user to click on a malicious link or execute some form of attached malware. Like emails, web-based driven-by-downloads also use social engineering tricks in order to attract the user attention. Web surfers are "invited" to click on web links that are apparently safe triggering the infection.

Two known paths to web-based driven-by-download infections are widely used by hackers. The first path is leaded by browser exploitations such as those involving cross-site scripting (XSS) and IFRAME browser vulnerabilities. Normally, such browser vulnerabilities allow malicious code to be executed in the browser's security context creating, in the end, an open door for malware installation. The second path requires additional doses of social engineering to trick the user into clicking, downloading, and installing the malware itself.

On the attacker, hacker or herder's point-of-view, Microsoft Windows server services such as Netbios and SQL Server are great sources of vulnerabilities exploited to compromise systems. Using week passwords in Netbios shares is a good example of a vulnerability that can be exploited by both hackers and bots through dictionary brute-force attacks. SQL injection exploitation is transversal to all database engines and is related to poor software design. SQL Server service, on the other hand, is known for its buffer overflow vulnerabilities. Slammer worm used such vulnerabilities in order to turn itself into the fastest known spreading malware of its time jumping into the news headlines, for example. [8]

[1] A *dropper* is a piece of software designed to download, install and execute additional software packages

2.2.2 Bot code installation or activation

Exploitation code and droppers are normally light and small sized components. More advanced features means heavier malware.

Beyond autonomous work in malicious activities and own survival in a hostile environment, Virus, Worms and Trojans, can upgrade themselves or the compromised system itself in order to allow controlled communications between all the bots and the herder. In this thesis context, installing or activating a malware so it can participate in a botnet and perform coordinated actions, is defined as *Bot Code Installation*.

When a communications network is created between two or more infected hosts and the attacker, a *Command and Control Channel (C&C)* is said to be established. The infected hosts become *Zombies* or *Bots* and the attacker becomes a *Controller*, *Botmaster* or *Herder*. The described network is said to be a *Botnet*.

2.2.3 Virus, Worms and Trojans

Although being used interchangeably when referring to malware, the terms Trojan, Worm and Virus have different meanings.

Virus is a computer program that attaches itself to a binary (infects it) and performs malicious activities. In order to perform its activities, the infected binary must be run by a user. In this way, virus infections and operations always involve user interaction.

Worms, on the other hand, don't require human interaction to spread their infection to other hosts or systems. They're a form of Virus in the way that they also perform malicious activities.

Finally, Trojans are different from the malware above in the sense that they don't spread any infection though being able to perform malicious activities such as file deletion. Trojans serve the same purpose as its wooden homonym that allowed the Greeks to enter the city of Troy in Greek mythology. When a Trojan is run by a user, a backdoor is normally created in order to allow remote access to the host running the Trojan.

Either way, the functionalities of Virus, Trojans and Worms are many times blended into a single malware application in order to improve its reproduction and damage capabilities.

2.3 Command and Control

There are many ways a herder can control its botnet and there can be found centralized, decentralized and mixed control topologies. [9][10]

On the centralized topologies corner, IRC and HTTP rule de Command and Control (C&C) panorama. Simplicity of deployment made these two protocols preferred by initial botnet programmers and designers. With simplicity, though, came easier botnet takedowns.

In this section it will be provided an overlook of several C&C topologies together with some C&C related methods used to improve a botnet's resilience.

2.3.1 Internet Relay Chat (IRC)

Internet Relay Chat (IRC) is a widely-used synchronous protocol used for real-time text-messaging in internet. It was designed having in mind group communications or conferencing. [11]

A simple diagram showing an IRC topology is shown in Figure 1.

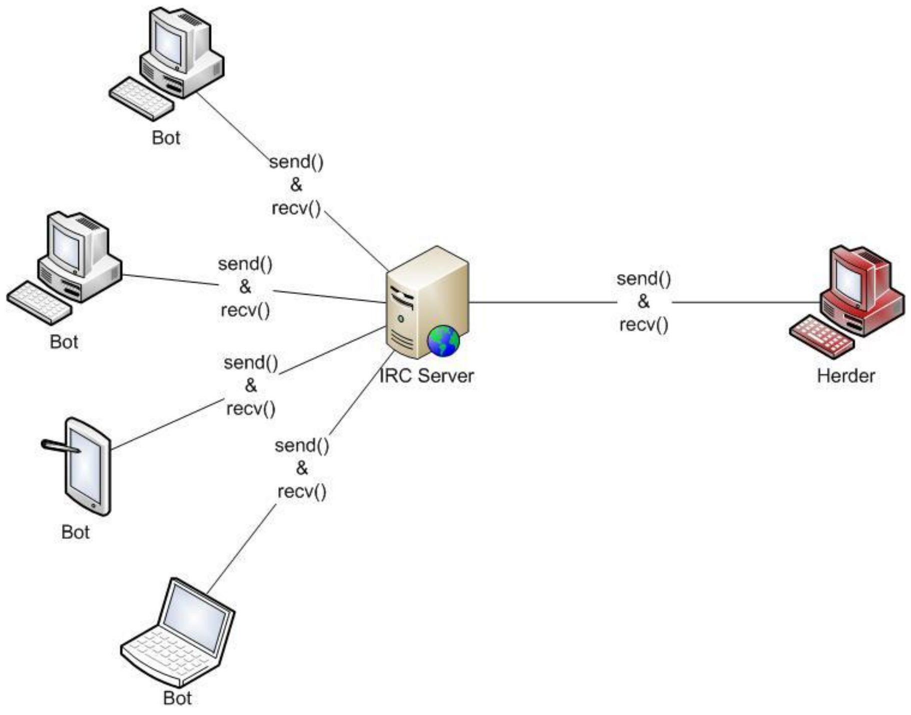


Figure 1: IRC Botnet Topology

In this C&C topology, all bots connect to the same IRC channel and wait for commands in the form of chat messages issued by the herder (typically a normal IRC user). In a small variant of this type of botnet organization, direct communications between bots have also been observed. As a result of the inherent protocol operation, the problem of command broadcast to multiple bots is simplified in this architecture since messages sent to a specific channel are viewed by all subscribers.

IRC topology is considered to be a centralized one since communications between all participants in the botnet (bots and herder) must exist in the same IRC server and channel.

One way of disrupting IRC botnet operations is shutting down the control server or the channel used to control the bots. A deep analysis of an IRC botnet behavior can be done by subscribing to the botnet channel and listening to messages exchanged between the bots and the command server.

2.3.2 Hypertext Transfer Protocol (HTTP)

In this C&C architecture, a basic web server acts as the command center to all bots' operations. On the bot's viewpoint, HTTP operates in *pull-style* mode [12]. Bots can pull orders from a web file published in the command server by the herder.

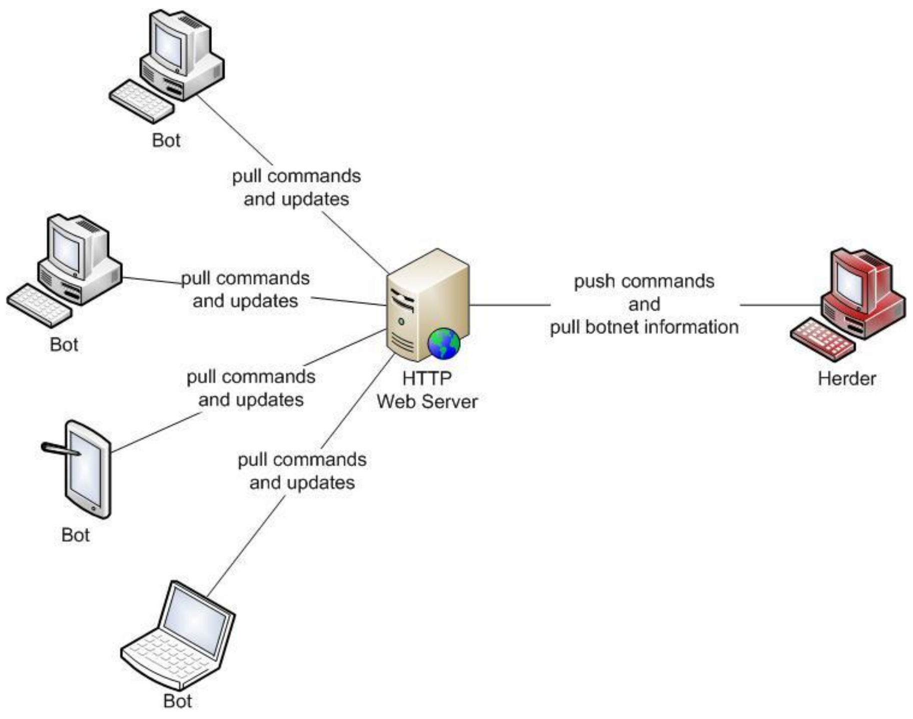

Bot
pull commands and updates
Bot
pull commands and updates
pull commands and updates
Bot
pull commands and updates
Bot
HTTP Web Server
push commands and pull botnet information
Herder

Figure 2: HTTP Botnet Topology

Unlike the C&C architecture of IRC botnets, this type of C&C forces bots to have some sort of synchronization method in order to start the pull messages process.

IRC protocol is synchronous in the way that every client (bot) subscribing a botnet channel receives commands in the moment they are issued. HTTP clients, on the other hand, pull data only when needed (asynchronously). In that way, completely synchronized bot actions are more difficult to trigger using an asynchronous-based C&C than using synchronous-based architectures.

To minimize lag between bot responses, bots must contact very often its HTTP command center. This timely behavior can be used to track bot existence in a network.

Variations of the HTTP architecture can be established in order to hide the herder and improve a botnet's resilience. One example of such variations is a topology where several HTTP servers are assigned the command of several sub-botnets. That way, the single point-of-failure that represented the unique web server is minimized. Taking down an intermediary control server only compromises part of the botnet. An image describing such topology can be found in Figure 3.

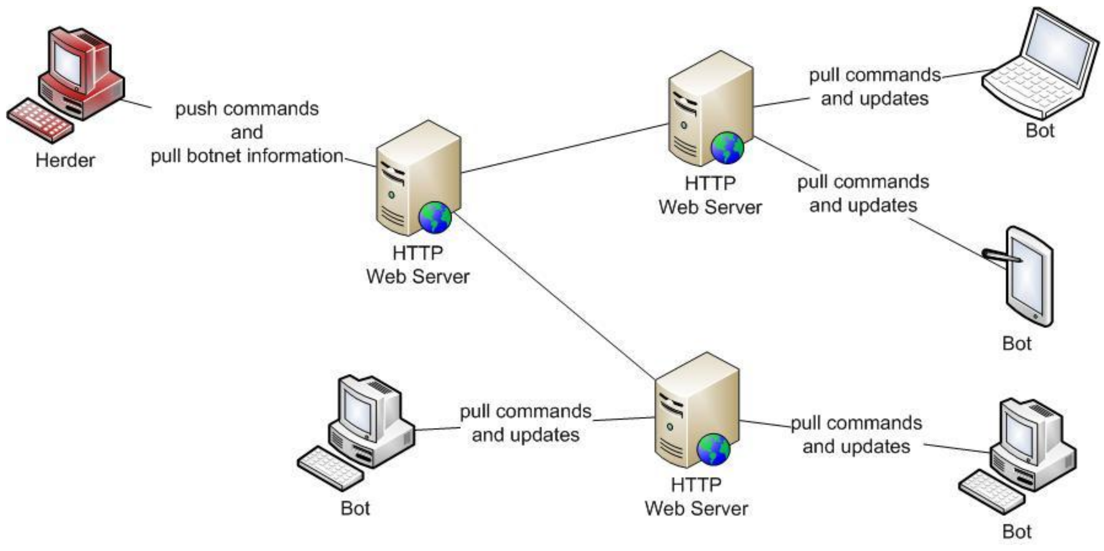


Figure 3: HTTP Botnet Hierarchic Topology

A similar but more resilient hierarchic HTTP topology can also be established by using bots as control servers. This kind of topology is also known as fast-flux [13], [14]. Hierarchic topologies can also be deployed in IRC-based bots as described in [4].

Unique servers acting as heads of the botnet were a big point-of-failure to botnet operations since they could also be attacked or blacklisted thus compromising partially or even entirely the botnet. Among other strategies, botnet designers turned their heads to Domain Name Systems (DNS) [15–17] in order to improve botnet resiliency.

One approach taken by botnet developers to harden botnets was to embed in bot code, one or more fixed domain names where control servers could be contacted. Domain records could be changed to new IP addresses if the control server became compromised. Although it's possible to blacklist DNS records, it's hard to do it efficiently on a large scale basis in non-controlled Autonomous Systems (AS).

HTTP-based architectures are simple and can be implemented in numerous ways including the use of free services such as Twitter that available in internet. Botmasters can use several twitter accounts to publish its botnet commands [18].

2.3.3 Peer-to-Peer (P2P)

In a P2P network, nodes are both clients and servers. In a content-based publish/subscribe-style communication, a node acting as a server can publish a file (for example) by using an identifier derived from a hash of the published file.

.

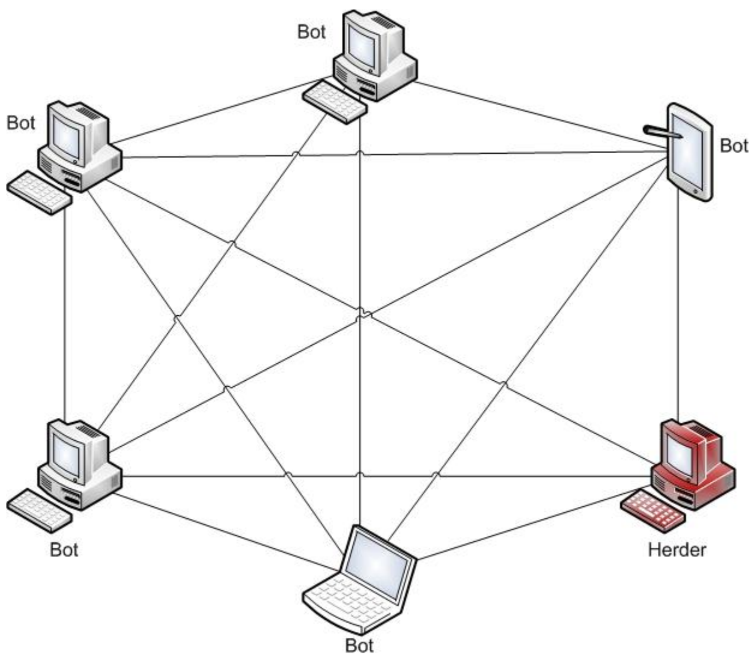


Figure 4: P2P Botnet Topology

Knowing the public hash function, client nodes can subscribe to the file published using the hash provided by the server node. P2P network system then searches for the file and delivers it to subscribed clients.

Many file-sharing systems such as Gnutella, eMule, or BitTorrent use unauthenticated communications. The unauthenticated file share behavior, used by all the popular file sharing systems, implies that any file received by a client node that matches the subscription is assumed to be correct [19]. Unauthenticated file sharing can be used to disseminate malware hidden in supposedly benign software.

In P2P file sharing systems the publishers and consumers information is almost inexistent. Nor the receivers know who's publishing, nor do the publishers know who's receiving. This property turns P2P file sharing systems attractive to botnet developers. [19]

P2P networks can be considered structured or unstructured. CHORD distributed hash table protocol (DHT) and Content Addressable Network (CAN) are examples of structured P2P topologies [10]. On the other hand, unstructured topologies can also be found in hub-and-spoke networks [10], for example.

P2P botnet communications design follows two orientations: using already implemented P2P protocols and using own P2P proprietary protocols. The Storm (Peacomm) Worm/Botnet is a good starting point for P2P bot communications analysis since it implemented both forms on different versions of the worm/botnet.

The first version of the Storm Worm used Overnet, a Kademlia-based P2P distributed hash table protocol (DHT). From October 2007, besides using Overnet, newer versions of Storm started using its own P2P network protocol. Though being identical to Overnet, the Storm P2P network improved resiliency by implementing XOR encryption of the communications. [19]

In a P2P botnet, malicious command file identifiers (keys) are generated using current date and other random variables. Sharing the key generation algorithm between the control servers and the hosts allow the herders to publish commands for the botnet P2P nodes (bots) to consume. [19]

Besides encryption, there are some other methods to improve a P2P botnet resiliency. Some P2P botnets, for example, limit each peer connections in order to hide knowledge of the entire botnet if a bot node is compromised. [20]

When open P2P file sharing systems are used to deploy a botnet it is very difficult to tell bots from normal P2P clients apart. Some properties of P2P can be used, though, in order to tell Traders (normal clients) from Plotters (bots) [21]. Among such properties we can find traffic volume, node persistence, peer churn[2] as well as regularity and periodicity. While traders normally download large files, plotters don't. Plotters (bots) use the P2P network exchanging small volumes of information. Moreover, since Plotters don't control network access, their connections tend to be more persistent and opportunistic in order to guarantee access to the commands issued by the botmaster. [21]

Peer churn can be used to distinguish Traders from Plotters since Traders have higher peer churn than Plotters. Plotters only try to maintain their connectivity. Finally, finding regularity and periodicity in communications is helpful in telling human from computer-driven actions. [21]

Disabling a P2P botnet is not an easy task but can be done. P2P botnets takedown normally involves pollution of the key space used by the botnet commands. In order to know the botnet P2P key space it is necessary, in first hand, to disassemble the bot's code and reverse engineer the algorithm behind key generation. This will allow the creation of botnet command file keys. The objective of a pollution attack is to flood the entire botnet P2P network with false or empty commands in order to disable installed bots. This is indeed a race against the botnet's herder to get the maximum command keys into the P2P network nodes. [19]

2.3.4 Fast-fluxing and Domain Name Systems for Improved Resilience

There are several ways DNS can be used in order to improve a botnet's resilience.

[2] Peer churn is the rate of a host's connections and disconnections from a file sharing network.

One method is the use of Dynamic DNS (DynDNS). Some DynDNS companies operate in countries difficult to reach by international law allowing herders to create and maintain domains even when abuse reports are issued. DynDNS allows herder to change their servers from country to country (and jurisdiction) making it very difficult to shut them down.

Fast-flux service networks take DNS usage one step further bringing redundancy and load-balancing features to botnet control channels. Fast-flux operations are based on two concepts: Round Robin DNS (RRDNS) and, the more advanced, Content Distribution Networks (CDN). [13]

RRDNS implementations are found when DNS servers answer "A" queries with several IP addresses for one given domain and the order of the IP addresses returned changes on every query made following a Round-Robin algorithm.

CDNs are very similar to RRDNS but its name servers are a bit more sophisticated. When queried, CDN name servers are able to return a list of possible IP addresses for a given domain based on client proximity to the IPs returned. Network Topologies, link characteristics and type of content requested are factors that contribute to the CDN "proximity" calculations. [13]

Botnets can take advantage of fast-flux servicing by deploying their own name and content servers on infected hosts. Beyond HTTP and DNS services, fast-fluxing techniques are also used to deploy SMTP, IMAP and POP services [22]. Compromised hosts can then act as name and email servers, participate in malware updates as well as deliver malicious content.

Fast-fluxing botnets can be implemented in two ways: single-flux and double-flux [22]. While single-flux networks use mechanisms such as RRDNS to constantly change the IP address of a domain, double-flux networks add an extra layer of protection by also changing the authoritative name servers responding a given request.

The purpose of deploying custom DNS servers is to restrict access to the correct control server. The process is done by deploying a list of domains and corresponding IP addresses to all the infected hosts through a C&C channel. This way, herders can point a well-known domain such as google.com to a malicious IP making very hard to black-hole the botnet communications. In addition, researchers and law enforcement authorities trying to find control servers by resolving the domain found in the bot code will be directed to different results appearing that a given domain doesn't exist.

Adding more layers of security and obfuscation to the system can be done using Domain Generation Algorithms (DGAs). DGAs are normally based on inputs such as the host system's date and some other random values. With a known DGA shared between bots and herder's control servers, control domains can be generated on a daily (or even hourly) basis allowing constant changes of the control channels. This procedure is also known as Domain Fluxing [13], [14].

In a real typical Domain Fluxing operation, several possible domains are generated daily or hourly. Herders only need to register some of them and install corresponding control servers. Bots then try to contact all possible control domains one after another until one working is found. Communication is then started with the server until new change in control domains.

Although being highly dynamic, fast-flux domain operation gives researchers the opportunity to control, totally or partially, a botnet. Once a DGA-based bot is captured and its code disassembled, the DGA can be reverse engineered. Knowing the DGA, researchers can register some domains ahead of the herders making it possible to control the botnet. This procedure has been taken in the Torpig botnet take-down described in [23].

Some efforts made to detect DNS techniques involve the regular check of DNS entries looking for domains having low Time-To-Live (TTL) values. Rogue and malicious related DNS entries have typically low TTL values in order to allow rapid changes propagation [15], [24]. In order to improve such detection approach accuracy, the number and change rate of A and NS records returned in a DNS as well as the presence of broadband or dialup networks is taken in account after low TTL DNS entries have been found. [22]

Though very evolved, flux node agents keep one old bot behavior by regularly contacting the control server to announce availability as well as check for updates [22]. Such behavior represents the Achilles heel of the botnet and can be used for detection purposes.

In the end, real botnets use interchangeably all described architectures to operate their malicious activities. P2P, IRC or HTTP can be used in the communications between bots and herders to exchange botnet management information as well as downloading updates and attack information (SPAM mailing lists for example).

2.4 Maintenance and Availability

Like all software, bots must have their bugs fixed and features upgraded in order to keep high availability as well as improve attack vectors. This section will present bot and botnet features not related to C&C that contribute for higher botnet availabilities.

2.4.1 Updates and Upgrades

In a maintenance and availability context, bot updates mean improved skills to survive hostile environments. Survival is a constant struggle between bots and its hunters. On the other side of the bot's barricades, security products evolve every day. Upgrades and updates (UU) are the only way a bot can keep up with security products evolution (as well as the other way around).

Updating and upgrading (UU) a bot are processes that can be triggered by both the bot and the herder through its control servers. Bots can start the process every time they contact the control server by checking the availability of updates and then downloading them. This is usually what happens in pull-type bots like the ones using HTTP topology architectures.

Herders can also start the UU process in push-type topologies like the ones in IRC botnets. To accomplish that, botmasters send commands to desired bots instructing them to update and, at the same time, indicating how to perform such task (download servers addresses, protocols, etc.). Updated binaries can be distributed using control servers, files shared in P2P networks, file-sharing websites, and others.

2.4.2 Deception

Longer malware lifecycles also rely on deception. By hiding their presence and activities in hosts and networks, bots improve their chances of survival. Besides hiding its presence, hiding its code and algorithms from researcher's eyes is also an important bot deceiving behavior.

Data encryption helps bots communications to keep its anonymity. Messages exchanged between bots and herders in clear text can be used to create very simple signature based Intrusion Detection Systems (IDS). Beyond the ability to help evade signature-based IDS's, encryption allows to hide bot commands from network sniffers.

Malware can also inject its code into system processes. Process injection is used by bots to elevate execution privileges in a host as well as hide its presence from firewalls and antivirus. Code injected into a system process is executed in the own system process security context evading in that way some firewalls policies as well as antivirus detection.

Code obfuscation and compression processes work like data encryption but on an executable level. Obfuscation and compression makes bot's code and algorithms harder to analyze.

Rootkits[3] are very often packed with bots. They hide the bot's presence from the host's users by hooking to system functions and hiding operating services queries results where the bot's trace could appear. Some of the first rootkits that appeared in Unix systems substituted the system "ps" tool by an altered version. The rootkit's "ps" tool simply returned all running processes in the system except those intended to hide.

Botnets can also be used as proxies in order to accomplish deception. Herders can use bot-based proxies' networks in order to hide activities and evade identification. [2]

[3] The origin of the term rootkit comes from initial Unix operating system ages. It is a concatenation of the word "root" (default super user account on Unix) and the word "kit" (as in package).

2.4.3 Malware wars

Besides antivirus, firewalls, IDS's and other security software, bots need to be aware of another menace to its integrity: other malware. What happens when an already infected host is infected by other malware? Perhaps biomimetic can help to find an answer.

An infected host in computer science is like a host in biology: an organism that harbors a parasite. In biology, different parasites can coexist and even help each other but they can also be opponents.

In infected computers, malware can act the same way as parasites in biology. If occasionally it is possible that certain malware even installs other "friendly" malicious programs, it can also happen that antagonist malware ends up installed in the same host. Some bots are prepared to work as real antivirus, detecting and removing other foe malware. This is the case of the TDL bot for example. [25]

Concerning maintenance and botnet availability, there are two very important statistics to the herders: number of active bots and bot lifetime.

2.4.4 Number of active bots

The number of installed bots is different from the number of available bots listening to commands in a state of readiness.

The number of available bots is related to hosts connectivity issues and time-zones. Bots installed on laptops and other mobile equipment not always connected to the internet result in intermittent availability and readiness of the bot task force. On the other hand, bots installed on distinct time-zones have different availability schedules since many hosts are turned off during the night.

A very high number of active bots is the objective of every herder. For that matter, many botnet's management consoles have features like IP geo localization and bot's time-zone information. These features allow herders to evaluate and correctly preview the entire botnet's readiness state.

2.4.5 Bots lifetime

Another important data for the herder is the malware lifetime. Host recruiting is a hard task and, has seen before in this section, good malware design is important in order to achieve longer malware lifecycles and maximization of the infection effort.

Well-designed bots are like strong and prepared soldiers which have a better chance to survive in hostile environments.

2.5 Attacks

After all the infection and survival efforts there comes a time when bots start executing the tasks they were designed for. These specific malicious tasks, known as attacks, can have their motivation in financial gains, personal pride, social, politic and religious causes, war, as well as simple access to confidential information.

In this section, known attacks are described and briefly analyzed.

2.5.1 Denial of Service (DoS)

Denial of service attacks involve successive malicious connections to a specific target host in order to reduce or disrupt its ability to respond to legitimate requests. DoS can become a Distributed Denial of Service (DDoS) if the source of malicious connections is spread across many hosts. [2]

In order to force a host or server to enter a Denial of Service state, the attacker can use a variety of well-known exploits such as ICMP and SYN flooding. Other forms involve access to a server in a brute-force manner. This can be achieved by instructing thousands of bots in a botnet to retrieve a specific webpage at the same time or get a specific file from a specific peer in a P2P network. Besides hosts and servers, DoS attacks can even target phones everywhere by simply flooding them with calls using compromised communications servers.

2.5.2 Theft

In the Theft Attack group we can find all sorts of information interesting enough to be stolen such as online banking credentials, credit card information as well as personal and industrial confidential information.

As DoS, theft can target users, governments, companies, and even security agencies using social engineering and other exploits already mentioned in the subsection related to "Initial Infection". Theft can be achieved by the herder through automated bot processes in charge of collecting browser cookies or hijacking user web sessions. Key loggers also allow bots to collect and send user's password information to the herder. Anyway, the goal of the herder is always access some, otherwise inaccessible, private information in order to achieve some form of gain (financial or not).

2.5.3 Click fraud

Ad programs such as the Google Adsense Program are becoming a more and more popular form of income in web businesses. Ad programs allow companies or users delivering web content to be paid when other users click on ads shown together with their web content.

Bots can be used to produce credible false user clicks on specific websites in order to artificially increase ad-based profits.

2.5.4 BlackHat SEO

A BlackHat SEO attack is capable of changing search engines results making malicious pages to show in the first positions of a web search. This increases the visits to malicious web sites and, thus, the chances of spreading malware and increase botnet host recruiting.

BlackHat SEO strategies can also be used in P2P networks. The attacker can deploy a malicious payload in a server and then instructs all its bots to subscribe that particular file. This action will make the malicious file to rank high in Torrents search engines producing the same result as the Web search strategy.

2.5.5 Spam

SPAM email is one of the biggest internet problems nowadays. The Symantec September 2011 Intelligence Report indicates that SPAM email reached the value of 74,80% of all email communications on that month. [26]

SPAM email is used to spread infections being an attack that leads the way for more attacks. SPAM is normally used to perform Nigerian scam types, distribution of malware and links to malicious websites as well as phishing scams.

2.5.6 Adware, Spyware, Scareware

Bots are also capable of adware, spyware and scareware installation.

Adware is a piece of software that shows unsolicited ads to the user. Besides the annoying fact of constantly popping-up windows to the user, Adware can itself promote links to malicious websites.

Spyware category can include key loggers, screen capture utilities as well as search for online credentials. Spyware can collect and send to the attacker user keystrokes and snapshots of a user's desktop as well as look for online credentials to services such as home banking websites and other payment services.

2.5.7 Brute Force

Brute Force attacks are related to password protected access to information or resources. Many network services such as Netbios, FTP, SMTP, and SSH are password protected services. Bots can flood a service with authentication requests until the correct credentials are found. To help in the process, botnets can use cooperative bot behavior to launch distributed brute force attacks as well as use dictionaries[4] and rainbow tables[5] to speed the success of the attack.

2.5.8 Gaming

Not very often mentioned in scientific works, game bots are used to play online games on behalf of real users. Real gamers can create game characters and ask the bots to play for them. Online games such as Travian and World of Warcraft are good examples since they need timely user interactions to evolve in the game. Bots can automatically do required actions for the gamer as well as cooperatively help the botnet "guild" and make it grow in the game itself.

Gaming bots have two main goals in mind: Pride and Profit. Pride comes when the gamer wins the online game (even cheating). Financial profit can also exist in online games since real marketplaces are available to gamers. Such marketplaces allow the exchange of virtual artifacts and characters for real money. Game character evolution can thus bring real profits to online gamers.

[4] Dictionary attacks give the attacker the ability to use a dictionary of normally used words for passphrases. This can reduce drastically a brute force attack when week passwords are used to protect services access.

[5] Rainbow tables work like dictionary attacks but use hashes of possible passwords. Passwords are often kept in a database not in clear text but hashed. Deciphering a hashed password using brute force is a time consuming task. Rainbow tables act as dictionaries between real passwords and their hashed values. This fact lights up the task of deciphering hashed passwords when the attacker is able to get them.

2.6 Botnet History: The Malware Chronicles

"If you know the enemy and know yourself, your victory will not stand in doubt."
Sun Tzu. The Art of War.

After basic aspects of botnet operations are known, some real botnets were analyzed in order to establish strong foundations for the work to come as well as relate some of the theoretical aspects discussed before to the real botnet world. The work done collecting all botnet information was needed since, at the time of this study, no simple, transversal and centralized study of major botnets could be found in scientific published researches. To confirm the data collected, information was checked between several long established security companies and entities as well as crossed with some of the few scientific papers published.

The study made on known, important and evolved botnets can be considered, along with the previous subsections and all scientific work referenced, one of this thesis cornerstones supporting behavior analysis and posterior dissection of all botnet operations. It has been fully disclosed in Appendix A.

Real botnet analysis can also be seen as the previous work done by a general when preparing for battle. All aspects of war must be known. As Sun Tzu pointed out very well, the knowledge of the enemy is primordial. Therefore, the study presented in Apendix A reinforces real botnet knowledge building a bridge between theory and existing botnets operations in real networks spread around the world.

All analyzed botnets were selected taking in account their specific capabilities and weight in malware dissemination and number of controlled hosts but many other botnets such as Akbot, Asprox, Bobax, CutWail, Gumblar, Kraken, Spamthru or others, also exist. It was possible to observe though that, generally, botnets not presented in Apendix A tend to use the same infection, spreading, maintenance and attack vectors shown and analyzed.

Important implementation variations and functionalities, though, were found in the Stuxnet and Zeus botnets. Stuxnet included PLCs to possible attack targets which was a behavior not seen previously in other botnets. On the other hand, Zeus was the first publicly known botnet framework to deliver a full bot and C&C server programming IDE allowing faster and custom botnet development.

It was also possible to observe and conclude that detection efforts need to be constantly updated in order to be effective. Both procedures taken by herders to upgrade bots and polymorphic versions of malware turn signature-based bot detection a difficult and challenging task.

Vulnerabilities patch timing lag was found to be an important aspect of all infections achieved by malware products. Stuxnet, for example, was known to exploit two Microsoft OS vulnerabilities identified by codes MS10-061 and MS10-046. First malware samples using such vulnerabilities were found in the months of April and June of 2009. Microsoft patched these vulnerabilities in August and September 2010 taking more than a year to respond. [27]

On a host level, botnet detection and mitigation can be made by antivirus or firewall products using signature-based methods to detect infections as they occur. Among the anomalous behaviors that can be observed and detected, connections on strange ports not frequently used are strong indicators of malware activity.

On a network level, it was possible to observe that botnet operations are evolving towards encrypted communications and proprietary ad-hoc networks many times based on existing P2P protocols or domain fast-fluxing. Command and Control channels are also being embedded on existing networks formed by bots. This behavior turns bots into command servers and vice-versa making C&C server's detection and takedown a difficult task.

Since bots are designed to hide their presence, attack behaviors turn out to be moments where bot presence is most noticed. This behavior is inherent to attack behaviors since attacks intend to change something in the otherwise "normal" environment. In the end, and at a network communications level, detection is possible upon three possible bot behaviors: infection and proliferation (including scan behavior), C&C communications and attack.

Fortunately, like all software, botnet development is made under many constraints such as human resources (developers, testers, etc.) as well as available time and money. This contributes to the existence of “unbalanced” bots and botnets. If some botnets have evolved skills in terms of maintenance and resiliency, others turn out to be strong in terms of attacks and proliferation. They all end up, one way or another, having at least one point-of-failure that allows researchers and law-enforcement agencies to achieve their takedown.

2.7 Conclusions

In this section major types of bot and botnet behaviors were analyzed. From proliferation and maintenance to attacks, a research was made allowing the organization of structured information that allowed better knowledge of botnet development, operations and strategies. The gathered information is useful in botnet characterization.

An important aspect of the gathered data is that botnet design is evolving very rapidly in all its development vectors: from infection, to dissimulation and attack capabilities. The ideal botnet is virtually undetectable, constantly updated and upgraded, with high proliferation resources, as well as capable of launching all kinds of attacks exploiting all known system’s vulnerabilities.

On a botnet detection point-of-view, several bot characteristics and behaviors are passible of being used in time-efficient detections (before attacks occur). Every bot or herder action, such as those related with C&C communications and bot updates, results in network traffic capable of being captured and analyzed.

3. Botnet Detection and Analysis

"The quieter you become, the more you can hear." Baba Ram Dass

Although the thought of "silence" in a noisy environment can be considered awkward, when there is very good knowledge of the "background" noise, "silence" can be observed. Different sounds can then be distinguished from the background like seagulls screaming in a noisy sea shore.

In this section some methods of detecting and analyzing bot and botnet presence in a network or host are explained.

3.1 Honeynets, Honeypots and SPAM Traps

Honeypots are vulnerable hosts with internet connections waiting to be infected by malicious herders or malware. Hosts are left vulnerable on purpose by researchers so malware can be captured for further analysis. Occasionally, honeypots are infected conscientiously by researchers using previously collected malware through SPAM traps for example. A network of Honeypots is known as a Honeynet. [4]

SPAM traps can be deployed using publicly available email accounts to collect malware sent through SPAM email. Malware collected directly from an email or retrieved from included links is then kept safely for later analysis and debugging.

Honeypots, along with SPAM traps, are the privileged methods of gathering malware binaries.

3.2 Signature-based Methods

Malware signatures can exist in numerous forms. The most common malware signature dictionaries can be found in antivirus or antimalware software.

Every malware code, after compilation, produces a binary (executable) that has unique hexadecimal string characteristics that differentiates it from other programs. It is like a car's license plate or a person's signature. After analyzing the malware collected in honeypots and SPAM traps, a stream of bytes is retrieved from the malicious binary and put in a dictionary along with a malware name. These streams of bytes become identifiers of malware.

Signatures can be extracted from other sources than malware binaries. Communications payload and protocols can also be used to create signatures. Example of application detection using analysis of payload streams can be found in [28]. Exchanged messages can also be used to create a signature. Number of bytes sent and received as well as time intervals between messages become like barcodes. [29],[30]

Other signature-based methods exist to detect IRC botnets. One of them is presented in [31] and is based on the knowledge that bot's generated IRC nicknames usually follow fixed rules for creation. In this case, IRC nicknames become the botnet signature.

Finally, Deep Packet Inspection[6] can assist by allowing the creation of email signatures (among other applications) using network traffic monitoring. Email signatures are based in contents, line spacing and character count [4].

3.3 DNS-based Methods

There are three ways Domain Name Systems (DNS) can assist in botnet detection. These methods are presented below.

3.3.1 DNS records analysis

In order to hide and rapidly change C&C server's locations, some botnets use Dynamic DNS services (DDNS) [24], [32], [33]. These services allow a botnet designer to change the IP address reference of a C&C server to the much more volatile domain name. Herders can then

[6] Deep Packet Inspection, also known as DPI, is a network traffic monitoring technique that allows the verification of network packets payloads (on application level) and not only its headers.

more easily change C&C server's locations and IP addresses without the need to inform the bots of such change. Bots can remain connected to the same C&C domain name. Herders just need to change the DNS record corresponding to the C&C server.

In order to use and keep such system's high availability, corresponding DNS records need to have low Time-to-live (TTL) values. High TTL values are responsible for longer DNS cache values. Low TTL values contribute to volatile DNS records. Such volatile records end up to force more frequent DNS records refreshes in network equipment and hosts contributing to lower time lag responses to C&C channels changes by bots [24], [34].

As seen, botnet detection can thus be based DNS records analysis. Short DNS records TTL values are indicators of volatile domain names and possibly botnet C&C servers. If such DNS records tend to change frequently, then it is a good idea to analyze such domains.

3.3.2 DNS queries analysis

Although the analysis of low TTL DNS records is a possible detection approach, it has been observed that many legitimate domains (gmail.com and yahoo.com for example) also use in its DNS records such low TTL values. In [24] an approach based on the analysis of recurring NXDOMAIN replies to DNS queries proves to be more effective than TTL value analysis.

Considering previous knowledge of botnet C&C servers' domain names, every request made to a DNS server in order to resolve such names can be considered to be originated in a bot and thus used for detection purposes.

3.3.3 DNSBL queries analysis

DNS blacklisting is used by system administrators to block malicious network traffic. DNS blacklists are widely available throughout internet and can be used to check the reputation of a determined domain system.

Herders, and even bots, tend to consult frequently such lists in order to find out if their C&C servers were detected and thus in the way of being compromised. This way, frequent, specific DNSBL requests from same hosts are strong evidence of botnet presence. [35]

3.4 Anomaly-based Methods

Considering this section initial considerations, anomaly-based methods try to distinguish the seagulls from the noisy background in the seashore. In a network traffic analysis context, unusual or abnormal events are sought in order to trigger alarms.

There are two main approaches to anomaly-based detections: one involving previous network traffic captures of what should be considered normal behaviors, and other involving the formal definition of normal or abnormal behaviors.

Although providing theoretical detection for zero-day threats, both approaches end up suffering from the same disadvantage: It is hard to prove that the expected normal behavior used as model in both approaches is really normal. If we consider that the first approach can be based on real traffic capture, existing abnormal behaviors in initial traffic captured can interfere in the definition of the systems normality compromising anomaly detections. Likewise, a faulty formal definition can endanger the posterior detection phase and is prone to become outdated with new network services appearing every day.

Since this thesis focus is on botnet anomaly-based detection, a deeper analysis of such methods is done in section 4.

3.5 Related Work

The analysis of botnet network behavior is a relatively recent research area but has already produced some interesting and coherent results. Some of these researches are presented in this section.

BotHunter [36] tracks communication flows between internal and external hosts correlating them to IDS events in order to effectively detect internal host malware infections.

BotSniffer [12] uses statistical algorithms to analyze topology-centric botnets and detect their hosts crowd-like behaviors. BotSniffer work was complemented with BotMiner [37], where a correlation between C&C (Command and Control) communication and corresponding malicious activity is also established.

BotMiner [37] introduces a new way of grouping communications filtering, at the same time, communications to top 100 Alexa.com websites in order to reduce the traffic under analysis. This strategy can fail in cases such as Twitter-based C&C channels. Discarding all traffic to Twitter will turn twitter-based botnets invisible.

In BotMiner [37] it was used real botnet traffic in order to evaluate its FPR. FPR, though, is evaluated based solely on injected known malicious network traffic. Although published detection rates are near 100%, there is no way of knowing for sure the existence of other botnets in the collected traffic. The total number of detections is also omitted in the research making it difficult to really sense the real accuracy of the proposed detection method.

In [38] three metrics are proposed to detect a botnet cooperative behavior: relationship, response and synchronization. In this work, IRC bots operations are studied and analyzed in order to identify anomalous IRC channel structures as well as synchronized traffic and similar bot response times. Similarly, Strayer manages to establish a correlation between inter-arrival time and packet size [39].

BotCop [40] analyses the temporal-frequent characteristic of flows to differentiate the malicious communication traffic created by bots from normal traffic generated by human beings. The work done in [41] contributes to the botnet detection research by establishing a distance metric between a pre-defined IRC botnet traffic model and passively collected network traffic (flows).

However, distinct approaches from the aforementioned botnet detection vectors exist: DNS traffic analysis is one of them. This detection vector was explored in [24], [16], [32] and by BotXRayer [17]. Both *Dynamic DNS* and *Canonical DNS Request Rates* (studied previously by

David Dagon) are analyzed and tested in [24]. In [16] the host's reaction to DNS responses was studied along with the implementation of a host-based bot detector.

Another important area of botnet detection research is botnet measuring and characterization. In [10] the authors made an important contribution to this research by presenting a model for botnet operation and size estimation. On the characterization side, Honeynets and Honeypots keep the lead in the bot hijacking processes allowing researchers to deeply study bot code and modus-operandi.

Veronica C. Estrada enumerates in [18] many existing botnet weaknesses capable of being exploited by network administrators in order to detect or disrupt their operations. This work also identifies some points of the failure in intrusion detection research.

In [21] P2P botnet detection based on high-level P2P network traffic characteristics such as volume of exchanged data, host persistence on the P2P network and peer churn is studied. A good analysis on the system accuracy is provided although values presented suffer from problems similar to the ones pointed out in the BotMiner [37] research.

Many different botnet detection approaches have been taken in recent years and authors are unanimous regarding the fact that best systems should be hybrid ones gathering the strengths of different methods.

Besides all the matters regarding detection approach there could not be found uniformity in the methods used to evaluate all the frameworks studied. The biggest problem found was the lack of detail in False Positive Rates evaluation.

Another area lacking scientific publication is real botnet studies. Useful and complete information regarding bot and botnet operations could only been found, almost always, in major anti-malware companies.

3.6 Conclusions

Like everything in life, every detection method has pros and cons, advantages and disadvantages.

Anomaly-based methods are known to detect zero-day menaces [42]. This is true considering that the goal of such methods is the detection of activities that deviate from known normal behaviors. This advantage, however, falls to the ground when high false positive detection occurs, thus turning alert management by a system administrator into hell.

On the other hand, signature-based methods allow simple detection of already known and analyzed malware. They have the advantages of performing detection with low computing overhead and low false positive detection ratios [42].

An ideal and complete detection system should use all known detection methods to create a hybrid approach reaching the best of each method. At the same time, Honeypots, SPAM traps and malware analysis could be used to collect and retrieve unknown malware characteristics and signatures. Finally, working side-by-side, an anomaly-based system could handle all the zero-day malware operations.

4. Botnet Anomaly-Based Detection

“The ten thousand questions are one question. If you cut through the one question, then the ten thousand questions disappear.” Zen Proverb

In order to build a botnet, bot masters (or herders) need to install some bot code in the zombie-to-be machines. This can be done initially by infecting the vulnerable machines with some simpler form of malware (virus or worm for example). After initial infection done by some exploit code, virus or worms are able to download and install more advanced software enabling the herder to start whatever malicious activities initially planned. But machine recruiting and bot installation don’t make a botnet per se.

Any bot master needs to control his army of bots in some way. A command and control (C&C) channel needs to be established in order to instruct the bots of their actions (scan, recruit, upgrade, attack and others). C&C channels can be created using either centralized (IRC or HTTP) or decentralized (P2P, unstructured or fast-flux networks) architectures [34]. In [10] it is confirmed that C&C is often the weak link of a botnet, although C&C-oriented botnet disruption isn’t always the best approach to take.

Although the use of well-known C&C channels is still preferred by botmasters on the basis of stability, new types of C&C channels are constantly being implemented in order to evade existing detection techniques. The use of Twitter and RSS feeds service are some examples [18]. Any new type of C&C will be invisible to simple comparisons with pre-determined models of botnet operation. The use of behavior analysis and correlation is, therefore, fundamental to correctly identify such dynamic and dissimulated botnets.

C&C analysis is very useful since it could also provide information regarding the location and identity of the botmasters but, in theory, every botnet activity besides C&C (scan, attack, etc.) leaves a trace in the network that, when correctly analyzed, can be used to reveal the botnet itself.

There are several traffic data sources that can be used to detect network anomalous usage and botnet activity. Among these we can find DNS Data, Netflow Data, Packet Tap Data, Address Allocation Data, Honeypot Data and Host Data [3].

In order to allow the proposed system to operate in large networks, this analysis will focus on the study of netflows as this kind of data is simpler and faster to process than other types mentioned. Honeypot Data and Packet Tap Data require heavier, harder and slower processing while Host Data analysis falls into the category of antivirus protection systems. Furthermore, many supervised networks already implement netflow logging which could, in turn, contribute to an easier implementation of the detection system proposed. The use of netflows is also a good starting point when dealing with traffic privacy issues. The major drawback of this approach is the loss of traffic characterization details provided by other types of data.

Several botnet features can be used in order to determine bot-related anomalous network activities: relationship, response and synchronization are some of them. Correlations can be established between these features by analyzing all visible hosts' activity in a pre-determined period and can be established in both vertical and horizontal vectors.

Vertical correlation uses the information captured from one single host activities in order to identify its malicious intents. Horizontal correlation, on the other hand, exploits the synchronized and similar behaviors of several hosts possibly belonging to the same botnet [18]. Vertical correlations can be established, in theory, through the detection of anomalous responses to known request/response pairs. Though being feasible, used alone, this method can be very sensitive and prone to produce high false positive rate.

Many flow-based correlations can be established, such as the number of bytes and packets per host, port and flow; the number of different IPs and ports contacted by a host and the host's flow interval and duration. These correlations are able to identify network scans, flash-crowd behaviors as well as other anomalous network events.

The netflow analysis approach method proposed in this thesis is similar to systems like BotMiner and BotSniffer but its time-efficiency goal really differentiates it from the others.

While BotMiner and BotSniffer algorithms focuses mainly on the attack phase, this thesis's approach starts its detection operation in the initial host infection, bot scanning phases and C&C communications among bots and herders.

Similarly to other botnet detection systems, the goal of the proposed analysis goes beyond the detection of single infected hosts to target detection of specific global network anomalies that can lead to the identification of related and coordinated hosts' activity relative to botnet operation.

4.1 Bot Behavior

In order to recruit more bots for a botnet, many zombies scan the network for vulnerable hosts. This behavior should be the first one to produce clear evidence of possible bot activity and is independent of botnet type (HTTP, IRC, P2P, or other).

Normally, scan activity targets few ports within a specific host always having in mind specific vulnerabilities. HTTP servers, per example, normally respond to HTTP, HTTPS and, possibly, FTP requests (three ports). Other much stealthier types of scanning can be found in the internet. The verification of single port connectivity on specific and targeted hosts (using a list shared by bots), is one possibility among many others. Scanning activity can thus be detected by monitoring host port and IP connection rates.

Other ways of distinguishing bot activity from normal network use involves the study of bot automate behavior. Bot C&C connections are normally established using fixed time rates [40]. Though some bots can make its C&C connections randomly, this can reduce the response readiness of the botnet. Fixed time connection rates can be detected by analyzing temporal characteristics of netflows such as flow duration or inter-flow pauses.

Another characteristic of bot automate behavior can be found in the similarity of its network flows in terms of size and number of packets.

4.2 Botnet Behavior

After identifying bot behaviors, it is important to establish the botnet behavior model. This can be achieved by observing the holistic characteristic of a botnet. When observed together, though independent, bots behave like a single intelligent entity. This synchronized and related behavior must exist so the botnet can be useful to the botmaster (from scan to attack activity). Similar and synchronized network traffic should thus be strong evidence of botnet activity.

4.3 Useful Traffic Data for Anomalous-Based Detection

Network traffic analysis is based on available data. In this section some useful data capable of being extracted from network traffic and used in anomalous-based botnet detection is identified and discussed.

4.3.1 DNS Data

DNS data comprehends DNS records and its enclosed information such as domain names, IP addresses and time-to-live values among others. DNS server logs including queries made to the server and DNS black lists (DNSBL) can also be included in this category.

4.3.2 Packet Data

Packet data includes the collection of raw network packets that cross a computer network. Datagrams are valuable sources of information since they include raw data exchanged in the communications between two peers. Packet data analysis can be used, for example, in signature-based malware detection systems and identification of specific application communications.

Packet data analysis can be made at different internet layers such as internet, transport or even application layer where real data payload transmitted can be analyzed.

Deep Packet Inspection is a technique widely used by security software companies and even country governments for network monitoring and control. The technology involved and its use have recently been in the center of the debate about internet anonymity and censorship. [43]

4.3.3 Flow Data

"A flow is defined as a unidirectional sequence of packets with some common properties that pass through a network device." [44]

Flow data is retrieved from capable network equipment's such as routers and switches and is the result of packet data aggregation. There are several protocols and aggregation algorithms developed and available.

Sflow is a protocol maintained and promoted by the sFlow Consorcium [45]. Sflow is implemented in innumerous network equipment but has not gained the relevance of Netflow or IPFIX.

Netflow was developed by Cisco Systems and has become a standard used by other network equipment manufacturers such as Juniper. Version 9 of the Netflow protocol is described in the informational document RFC3954 [44] and was the basis for the Internet Protocol Flow Information eXport (IPFIX). IPFIX is described in RFCs 5101 [46] and 5102 [46] and, at the time of this writing, the industry standard although the majority of network equipment's installed still use the Netflow protocol for packet aggregation.

Besides network equipment, there are several other tools that allow packet aggregation in Netflow format. One of the these tools is netprobe, part of the ntop tools suite [47]. Nprobe can be installed in a linux host connected to the mirrored port of network equipment. Nprobe collects all packets received on the mirrored port and creates netflows. Netlows can then be collected and saved to disk using the nfcapd tool (part of the nfdump suite [48]).

Netflow aggregation is normally done on source IP address, destination IP address, source port for UDP or TCP (0 for other protocols), destination port for UDP or TCP (type and code for ICMP and 0 for other protocols). Other flow information such as start and finish time of the flow, number of bytes and packets, Type of Service (ToS) value, as well as TCP flags for TCP flows can also be included in netflows.

Although being the aggregation result of more specific information, netflows can assist in characterizing communications between hosts thus helping in network anomalies detection efforts.

4.3.4 Hosts Address Allocation and Geographic Location

Although being inferred from other data sources such as Packet and Flow data, address allocation and geographic location are important to model botnet propagation through time zones. [37]

Host localization is important in the way that it can model botnet global activity and bot readiness states. Hosts around the globe are turned on and off crippling a botnet's overall capacity. Knowing bot's IP allocation and localization is important to preview a botnet true attack capacity.

4.4 Conclusions

Both bots and corresponding botnets reveal network behaviors capable of being analyzed and dissected in order to create a capable botnet detection framework. The selection of traffic data sources for detection purposes can be supported by a twofold classification of traffic data using *objectiveness* and *simplicity*.

Using an *objectiveness* quantifier one can grade traffic data capacity on providing clear and unambiguous information. On the other hand, *simplicity* can depict traffic data handling easiness. The *simplicity* quantifier includes manageability of the amount of data as well as computation speed needed for detection purposes.

Using a one-to-five integer classification range for *objectiveness* and *simplicity*, a classifier table can be built. In the *objectiveness* scale, one means low objectiveness and five very-high objectiveness. In the *simplicity* scale, one rates a complex handling of data and five an easier handling of data.

Packet data contents can be very objective since no more useful data can be added to the raw data collected. But dealing with raw data is not an easy task even on a small dimension network. Ultimately, raw data means all files, images, webpages and communications that a given organization produces or consults. Storage and analysis of such data is very difficult. Packet data is thus classified in the top right corner of the proposed Classification Table shown below in Table 1.

On the other hand, Flow, DNS, Address Allocation and Geographic Location data are simpler to use but provide less objectiveness since the useful information available is a scarcer.

Flow data, for instance, is much easier to analyze than packet data since it drops all raw content information aggregating only certain properties of raw network data. Although much less objective, flow records are resumed representations of packet data making them easier to work with than packet data. That way, flow data shows up with better balance between *objectiveness* and *simplicity*.

		Objectiveness				
		1	2	3	4	5
Simplicity	1					**Packet Data**
	2					
	3					
	4	**Address Allocation and Geographic Location**	**DNS Data**	**Flow Data**		
	5					

Table 1: Network Traffic Data Source Classification

Data sources shown can all be used in complementary ways to increase data *objectiveness* and corresponding useful information for detection systems.

5. Proposed Model and Prototype

"Nothing is more difficult than the art of maneuvering for advantageous positions." Sun Tzu.

Having gathered and organized all possible knowledge regarding botnet operations and detection, this section now presents a possible alternative detection approach as well as the considerations taken into account to support the proposed model.

This is, according to Sun Tzu, the most difficult part. All collected information needs now to be taken carefully into consideration and turned into real coordinated actions that, in the end, can lead to victory. In this thesis context: detection.

5.1 Initial Research

The Initial Research subsection presents all the work done that led the investigation to the final detection prototype proposed in this thesis. Initial research was made having in mind two goals: promote an early contact with the botnet detection paradigm and useful analysis tools as well as write a scientific paper for the 10th European Conference on Information Warfare and Security (ECIW'11). The acceptance and presentation of the paper on ECIW'11 was a motivational tonic for the work to come.

This subsection also presents the first botnet detection approach taken in the research being somehow useful in the way that explains the starting mindset alerting to some of the roads not to take in this kind of research.

In order to develop and test the initially thought heuristics, several tools were used: Nfdump project tools [48] allowed the capture and initial flow parsing; Microsoft SQL Server [49], Analysis Services [50] and Reporting Services [51] provided flow storage, processing, charting and statistical analysis; and, finally, BotAnalytics (developed in C# during this research) allowed the import of all collected netflows into an SQL Server database.

To allow the injection of the bot traffic (only available[7] in pcap dump format) into the netflow database, a feature was implemented in BotAnalytics that permitted the conversion of dump capture files into flows and corresponding insertion into the SQL Server Database. This conversion was made by aggregating captured packets into a unidirectional five tuple record (Source IP, Source Port, Destination IP, Destination Port and Protocol) with corresponding number of packets, bytes, start and end time of the flow. A flow aging mechanism was also introduced in the pcap dump import feature that automatically determined the end of a flow when, for an active five tuple record, a packet wasn't seen for, at least, sixty seconds.

The (initially assumed benign) traffic datasets used in this research were captured in the University of Minho network edge. They were collected and parsed using nfdump tools before being imported into the SQL database by BotAnalytics (internal proxy and DNS servers traffic was excluded at this time for the sake of analysis simplicity). Nfdump tools were installed in a server connected to a mirrored port of a switch at the edge of the campi network. All traffic entering and leaving the university's network was captured.

All the initial heuristics validation and development were made on a specific dataset representing the University of Minho UDP and TCP traffic on the 11th January 2011. More than thirty million flows were imported into the SQL Server database.

5.1.1 Scan Detection Heuristic

Scan detection is a common feature in IDS systems. Many possible detection approaches are possible to be implemented using statistical flow analysis. The goal of scan detection in the context of botnet activity detection is the identification of suspicious hosts that can later be observed in more detail applying, cumulatively, a wider set of heuristics in order to identify a botnet and its constituent hosts.

[7] Pcap Botnet traffic repositories freely available in the internet were very small and not trustworthy being dropped later from this study.

Modeling scan behavior was the first step taken to develop detection capable heuristics. Scan activity flows were assumed to have small packets (few bytes) with high number of destination ports and IPs involved within a relatively short period of time. In order to verify this behavior on the captured traffic, for each source IP and ten minute period, the number of distinct IPs and ports were counted. The result was then filtered using the *Distinct Destination IPs (DDIP)* and *Port/IP Ratio (PIR)* thresholds. PIR is calculated by dividing the number of distinct ports per distinct IPs contacted. These criteria preserved only flows with high distinct destination IPs and ports contacted. The result was then filtered by a *Bytes/Packets Ratio (BPR)* threshold, aiming to keep only traffic with small packets involved. The chart presented in Figure 1, shows the number of distinct source IPs with scan-like activity. It was built with a BPR of 100 bytes, and both DDIP and PIR of 5.

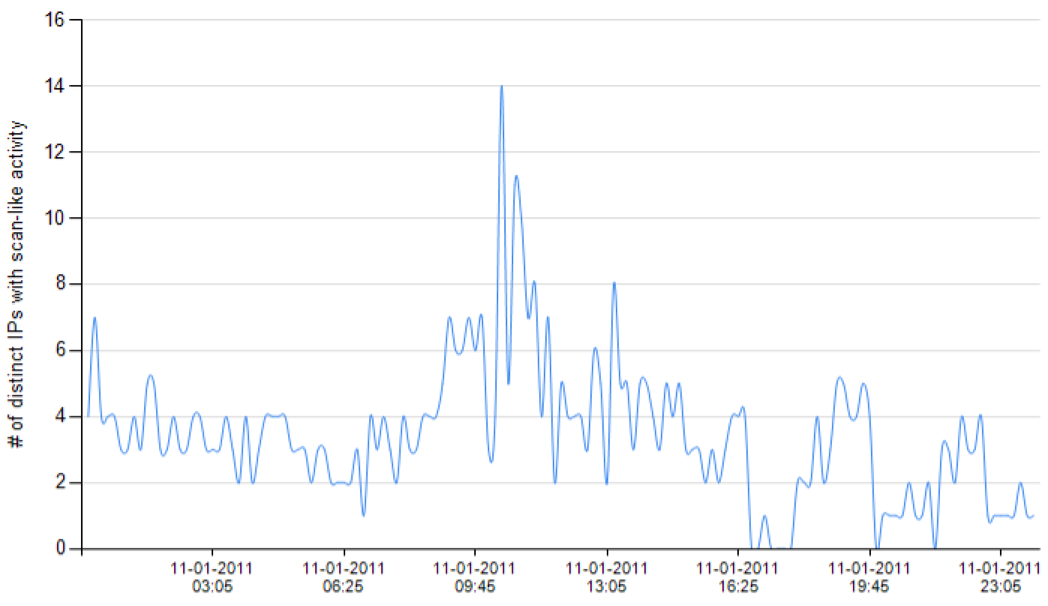


Figure 5: Number of distinct hosts with scan-like activity

The development of this first scan heuristic allowed to get a closer contact with the problem but didn't brought forward, per se, anomalous activity. Better results were achieved when aiming the study at specific application traffic as seen in the last heuristic developed for the initial research.

5.1.2 Crowd-like Behavior Heuristic

The crowd-like behavior heuristic was developed with the following observation in mind: flows with the same number of bytes and packets sent from many distinct hosts, during a relatively short time interval, are possibly related and very alike to what a botnet activity would produce.

Based on this assumption, and for each ten minute period, all the UDP and TCP flows with less than 1000 bytes and with the same number of packets and bytes where grouped. At this stage, every source IP with only one packet in the cluster was removed in order to filter out IPs with small contribution to the cluster. The 1000 bytes limit selection was based on the assumption that botnet communication is based on small packet exchanges [39]. Large bandwidth is commonly associated with bulk transfers or downloads. The resulting clusters thus contained hosts with similar flow characteristics.

The next step was to eliminate the clusters having less than five IPs. This threshold criterion was assumed after analyzing the resulting clusters content. Port usage analysis made clear that a vast majority of its flows were related to NTP and DNS usage making such clusters not important to this analysis.

Some clusters were then excluded based on their relevance. A cluster was considered relevant if its information was useful. Clusters with the majority of the source IPs seen in the ten minute period could not be considered relevant since they contained probably normal traffic. For this matter, an *Environment Ratio (ER)* and *Threshold (ET)* were defined. ER was determined by the ratio between distinct hosts in the cluster versus the total distinct hosts in the ten minute period under analysis. The ET used in this initial research was 0.001 and it was selected after carefully analyzing the clusters created and verifying again that clusters having ER higher than the selected ET (0.001) represented normal traffic (DNS, NTP).

Figure 2 shows a chart representing the number of distinct source IPs (size of the bubbles) with filtered flows having the same number of bytes and packets between 12:05 and 12:15 of the 11th January 2011. Two quick remarks can be done whilst observing the chart: no flows could be found with less than 378 bytes and more than 8 packets, and there were large clusters in the bottom left corner of the chart revealing the existence of many hosts with similar flows having few, small packets.

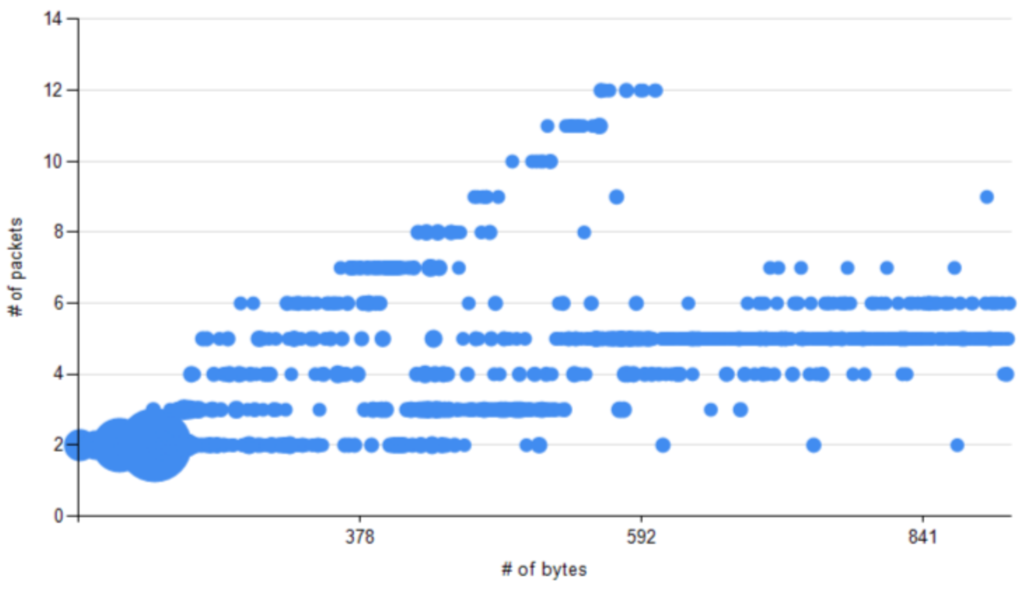


Figure 6: Distinct source IPs per number of bytes and packets

Finally, and for a ten minute period, a *Clustering Score (CT)* was determined for each IP. CT was calculated by counting the number of crowd-like clusters an IP belonged to. All the IPs with a CT below 5 and, thus, with less crowd-like behavior, were filtered out at this stage. All the thresholds and corresponding values here presented were used mainly for model and heuristic validation. The chart in Figure 7 shows us, for each ten minute period, the number of distinct source IPs belonging to suspicious clusters. Each of these hosts thus possessed crowd-like behavior.

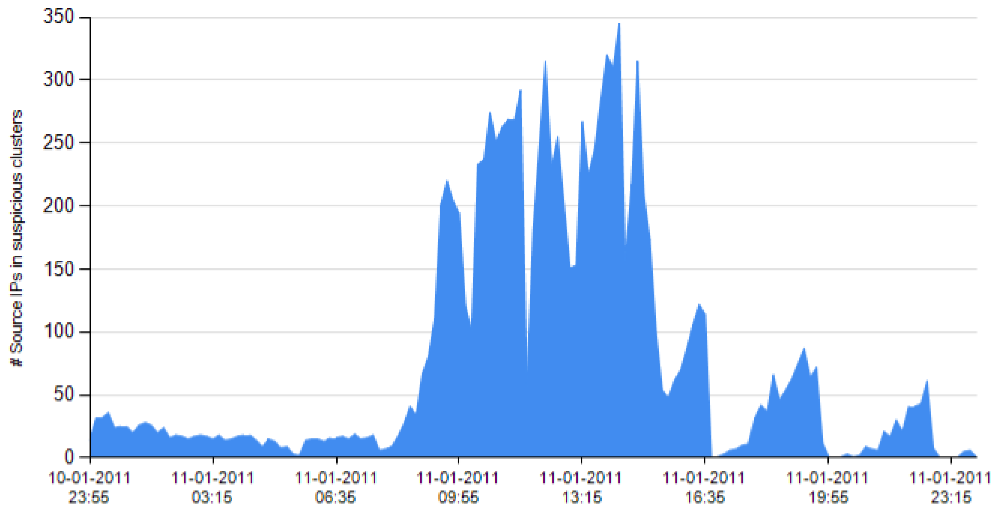


Figure 7: Number of distinct Source IPs with crowd-like behavior per ten-minute period

This data brought forward some interesting events when drilled down. The majority of the hosts detected with crowd like behavior revealed a high connection rate on unusual ports and with IPs scattered along geographically dispersed IPs. Further investigation was necessary to distinguish if such anomalous flows represented malicious activities.

5.1.3 Crowd-like SQL Server Scan Behavior Heuristic

The last heuristic proposed in the initial research was a much more targeted one since it was directed to a specific protocol and application behaviors analysis. Database servers are a very important source of application vulnerabilities. This makes Microsoft Windows SQL Server port scans very common. An existing buffer overflow bug in Microsoft SQL Server turned this application into the spreading mean for the Slammer Worm for example.

The goal of this analysis was to model specific scan behaviors. To find out if - and how - SQL Server oriented scan activity was being done in the campus' networks, all the flows not directed to SQL Server ports (1423 and 1433) or with more than 1000 bytes, were filtered out. Note that according to university's security policy it would be anomalous the existence of SQL Server connections with hosts outside the university's network.

Several such scans were found on the 11th of January 2011. The chart in Figure 4 shows the number of distinct IPs with connections (or attempts) to SQL Server ports related to the average bytes per packet used in the connection, during the day. It was possible to observe that most scan traffic had small bytes per packet ratio (BPR) and that this ratio didn't go beyond the 300 bytes.

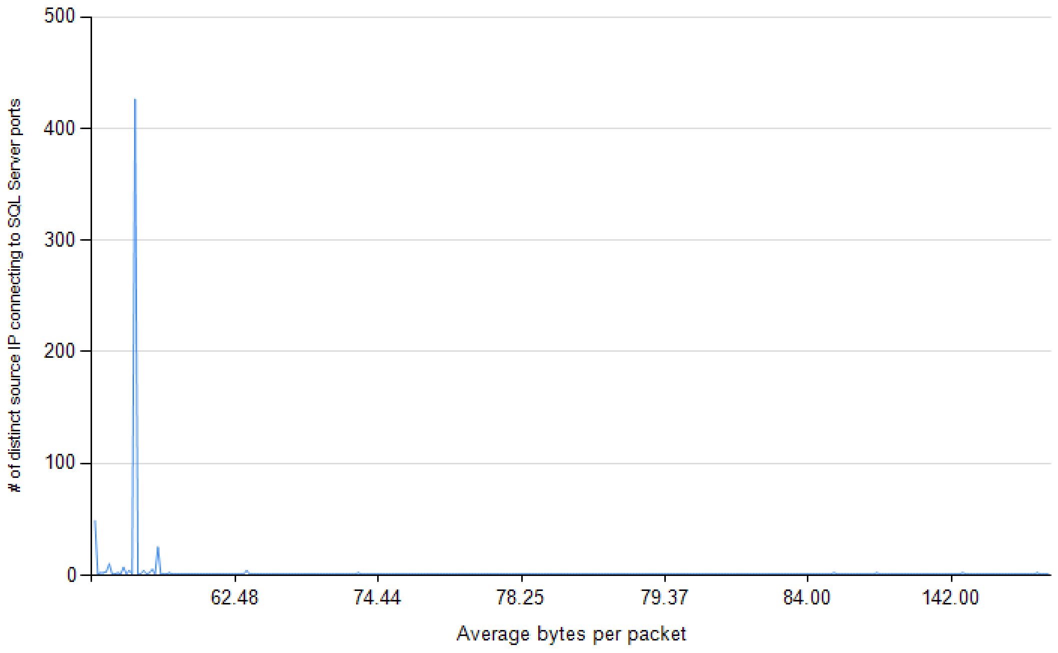


Figure 8: Number of distinct hosts connecting to SQL Server ports per bytes per packet

Bytes per Packet	Number of Distinct source IPs
48	426
40	49
52	25
44	10
46	7

Table 2: Top 5 bytes per packet clusters with higher SQL Server scan activity

The following step was to analyze in detail each host involved in the scanning, trying to model scan activities and, at the same time, detect crowd-like behaviors. Three important scan methodologies were found in the analysis of this traffic dataset: aggressive (ten thousand IPs scanned in ten minutes by one single host), constant (nine IPs scanned throughout the day in each ten minute period by a single host) and distributed (more than forty hosts scanning a different sets of IPs with a rate of one IP scanned per five minute period).

A crowd-like behavior could be established in the distributed scanning. Besides having a time-related correlation (five minute fixed scanning interval, all of the hosts were geographically close and scanned different sets of IPs. This final characteristic was the final proof of their cooperative behavior.

5.1.4 Initial Detection Model

The botnet detection model initially thought, but not implemented in a prototype, was to be based on two vectors: scanning activity and crowd-like behaviors analysis. Suspicious hosts should be first identified by the scan activity vector so they could have their network behavior thoroughly analyzed. Botnet activity identification should be made whenever synchronized and similar flows were exchanged by many different suspicious hosts.

5.1.5 Conclusions

The initial research presented in this subsection allowed addressing the problem of datasets size as well as getting a deeper knowledge of the available tools used to work with traffic data.

It was easy to observe that the tools used in this initial research were fare from optimal concerning computing efficiency. Only nfdump tools revealed the robustness and capacity to deal with large datasets of traffic data. For that matter, posterior research focused on nfdump tools, nfsen [52] and the Perl programming language [53].

In this initial work, though, it was clear that it was possible to detect anomalies through netflow analysis.

5.2 Netflow as the Traffic Feature Source

As seen before in section 4, several numerical properties available in network flows, can be used in order to detect botnet activity. Basic attributes of most netflows formats include flow start and end time, source IP and port, destination IP and port, bytes and packets in the flow, as well as the protocol used. Additionally, netflow is a standard for many network equipment vendors and its data is easier to deal with than packet data providing a good balance between *objectiveness* and *simplicity*.

Having in mind the *objectiveness* and *simplicity* provided by flow datasets, a decision was made to use this data source as preferred for prototype development. The goal to design a scalable prototype capable of operating under heavy network traffic helped in this decision.

Regarding privacy issues, the use of flow data also takes an important role when monitoring network traffic.

5.2.1 Netflow Attribute Analysis and Clustering

All the netflow attributes mentioned before can be used to cluster collected netflows. This can be made by basic attribute clustering or by composed attributes clustering. Some examples of composed attributes are the bytes per packet, bytes per source host, or the flow time interval by host, among others. More complex clusters can be made by aggregating basic flow attributes.

Created clusters can then be used to identify normal or abnormal traffic. Classification precision and threshold definition play, at the clustering phase, a critical role concerning the detection efficiency, particularly false alarms miss detection.

Several classification and clustering methods can assist in the clustering task. K-means and X-means [37], Cosine distance, Euclidean distance, Earth Mover's distance (EMD), Kullback-Leibler asymmetric distance [7], term frequency / inverse document frequency (TF/IDF), J48 decision trees, Naïve Bayes, Bayesian Networks [41], Entropy [54], as well as the entropy-related Similarity and Diversity Indexes are some of them.

5.2.2 Netflow Attributes Selection

Netflow records have many attributes available that are not currently important for this thesis approach. Among these attributes we can find the ones related to Multiprotocol Label Switching (MPLS), for example.

Besides attributes with short relevance for this study, other netflow attributes such as those related to IPv6 information were not used for the sake of simplicity, but should be considered in future researches.

For that matter, only the netflow records attributes in the table below were considered for the botnet detection prototype developed in this thesis.

Netflow Attribute	**Description**
Destination IPV4 Address	Destination host of the communication depicted by the flow
Source IPV4 Address	Source host of the communication depicted by the flow
Destination Port	Destination TCP or UDP Port of the communication depicted by the flow
Source Port	Source TCP or UDP Port of the communication depicted by the flow
Protocol	TCP, UDP, ICMP or other
Number of Bytes	Total number of bytes sent from source host to the destination host
Number of Packets	Total number of packets sent from source host to the destination host
Flow Start Time	Start Time of the flow
Flow Finish Time	End Time of the flow

Table 3: Netflow Traffic Attributes Selected

5.3 Traffic Attributes Heuristics

In this section all the netflow attributes and their heuristic exploration are explained in detail.

Since one of the thesis goals was to, if possible, define a prototype capable of operating under heavy network traffic load, the selection of a simple mathematical model that would, in turn, provide simple operating algorithms was necessary. For that matter, implemented anomaly score calculations were based on Simpson's Diversity Index [55]. Simpson's Diversity Index is widely used in ecology as a measure of biodiversity of a habitat. In this thesis context it was used to reveal diversity or similarity in the connection properties of hosts.

5.3.1 Simpson's Diversity Index

In "Measurement of Diversity" [55], Edward H. Simpson defines diversity as

$$D = \sum_{i=1}^{S} {p_i}^2$$

Equation 1: Simpson's Diversity Index

with S being a species richness or density and p_i the probability of a given specie appearing in an analyzed sample. In this definition, diversity values would range from maximum diversity (zero) to no diversity at all (one).

Another way of representing Simpson's Diversity is using its inverse form depicted in the equation below. In this definition, values are inverted from the previous equation therefore ranging from maximum diversity (one) to no diversity at all (zero).

$$\widetilde{D} = 1 - D$$

Equation 2: Simpson's Diversity Index (inverted)

In this thesis context, netflow attributes are considered to be species under analysis and inverse diversity values are used to determine anomaly scores.

5.3.2 Destination IPs Contacted by Source IP

In the *Destination IPs Contacted by Source IP* heuristic, two netflow attributes are used: Destination IP Address and Source IP Address.

Scan behaviors, for example, can be detected using this indicator. It could be considered to be anomalous in a network environment one single host contacting thousands of distinct hosts during a small period of time. On the other hand, low diversity in the number of contacts to different hosts can also be indication of anomalous behaviors. In a P2P network this can also reveal peer churn thus being an indicator of Plotter's (bots) activities. [21]

The proposed detection model calculates the inverse Simpson's diversity index ($\widetilde{D}$) for each source IP. In this heuristic, the number of connections established with destination hosts is considered to be a species differentiator. One source IP contacting several different hosts exactly once, for example, would be considered to have a destination IPs $\widetilde{D}$ of zero.

5.3.3 Number of Distinct Source and Destination Ports (by protocol and destination IP)

In this heuristic, four netflow attributes are used: transport protocol used (TCP, UDP), destination IP address, and both destination and source ports.

A high number of different ports contacted globally, or on a specific destination host, can be used to detect scan-type behaviors.

This heuristic can also be used to differentiate host and server activities. High source port diversity is normally seen in hosts since connections are started on variable source ports. On the other hand, servers tend to use mostly a fixed port normally related to the service provided.

5.3.4 Unidirectional Flows of Data

In order to find high failed connection rates, flows without returning corresponding flow records can be analyzed. Hosts with low ratios of normal bidirectional flows versus unidirectional flows are probably scanners. Although this heuristic is prone reveal scanners it must be used with caution since it can also indicate P2P file sharing hosts [21].

5.3.5 Bytes per Packet (BPP) and Packets per Flow (PPF)

Bytes per Packet (BPP) and Packets per Flow (PPF) are examples of flow basic properties aggregation. Values can be calculated using the number of bytes and packets in a flow divided by the number of flows under analysis.

Since botnet communications tend to be small sized [39], the BPP value is important and can be used to filter out bot communication patterns. Small packets communications are also observed in P2P Plotters activity [21] which helps in the implementation of P2P botnet detection.

Low BPP and PFP diversity values can potentially indicate repetitive network behaviors possibly related to botnet operation.

5.3.6 Time Between Flows (TBF) and Flow Duration (FD)

These heuristics are calculated using both the start and end time basic flow attributes. FD value is calculated subtracting the flow’s start time from its finish time. TBF, on the other hand, represents the lag between two consecutive flows from a given host. Both heuristics are calculated for each source IP/host.

As described in [21], timely behaviors can be used to distinguish human from bot activity.

In order to make this heuristics usable, normalization was needed. For that matter, different ranges were established for TBF and FD so normalized values could be used. Normalization ranges were defined in one hundred milliseconds intervals. Any heuristic value in one determined one hundred milliseconds range was assigned a “one-hundred-rounded” value.

Another important aspect of normalization is that it can reduce errors introduced by the netflow collection system. That way, similar values recorded in netflows are clustered together.

5.3.7 Horizontal and Vertical Analysis

Heuristics analysis can be implemented twofold: vertically and horizontally.

Vertical heuristic analysis use the netflow attributes observed in a single host’s communications [18]. In this category can be included attribute analysis such as the Time Between Flows (TBF) and Unidirectional Flows of Data. These analyses applied to the flows of a single host help the identification of malicious behaviors.

On the other hand, all inter-host’s study is considered to be horizontal heuristic analysis [18]. This analysis is useful in the way that similar heuristic values observed among distinct hosts are indicative of crowd or cooperative behavior.

In the proposed framework, horizontal analysis is achieved in a second-pass detection run on the collected netflows, and corresponding calculated heuristics, to determine possible crowd behaviors. This second-pass delivers additional anomaly scores and, in the way, identifies clusters of possible bots. Bots are, thus, identified by similar flow communication patterns or fingerprints.

5.4 Communications Fingerprints

Using the defined heuristics is then possible to create real communications fingerprints. Such fingerprints can be used to cluster similar network traffic patterns.

In this thesis fingerprint definition, every heuristic defines a characteristic of the analyzed communications between two hosts. A communication fingerprint is obtained by calculating the Simpson's Diversity Index (D) (or its inverse) of each of the following netflow heuristics: Distinct Source Ports (DSP), Distinct Destination Ports (DDP), Bytes per Packet (BPP), Packets per Flow (PPF), Time between Flows (TBF) and Flow Duration (FD).

Fingerprints are collected only from TCP and UDP network flows. That way, for each Source IP and Destination IP pair, there will be two fingerprints collected: one for TCP communications and another for UDP (similar to fingerprints from left and right pointing fingers).

In order to differentiate the lack of diversity ($\widetilde{D} = 0$) in the analyzed heuristics from the inexistence of network traffic using a specific protocol, inexistent fingerprint characteristics are identified by the value minus one (-1).

To better understand the fingerprinting model, a visualization scheme was developed. Some examples are shown in Figure 9 and following. It is possible to observe in the hypothetic example (Table 4 and Figure 9) that Source Port diversity is very small. This can be an indicator that the analyzed host is probably a server (or is acting like one).

Fingerprint Characteristic (Heuristic)	TCP	UDP
Bytes per Packet (BPP) Diversity Index	0,83	0,76
Packets per Flow (PPF) Diversity Index	0,98	0,12
Time Between Flows (TBF) Diversity Index	0,77	0,49
Flow Duration (FD) Diversity Index	0,56	0,71
Destination Port (DPort) Diversity Index	0,87	0,76
Source Port (Sport) Diversity Index	0,02	0,00

Table 4: Sample Communication Fingerpint Values

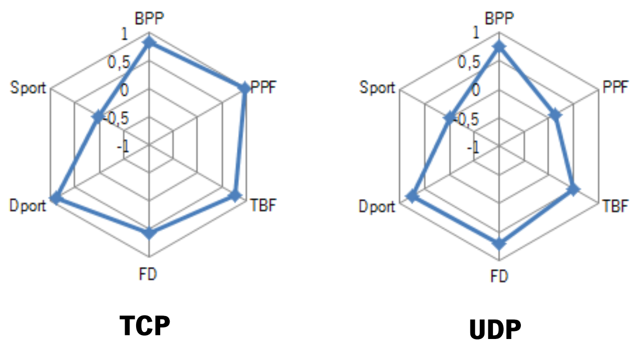


Figure 9: Communication Fingerprints (example)

To better understand the visualization model, two sets of fingerprints retrieved from hosts performing malicious activities in the University of Minho network are shown in Figures 10 and 11.

The example in Figure 10 was retrieved from real traffic and refers to a simple scanner connection. No TCP traffic was observed between the analyzed hosts. The only UDP traffic attribute with slight diversity value was related to the Time Between Flows (TBF) heuristic.

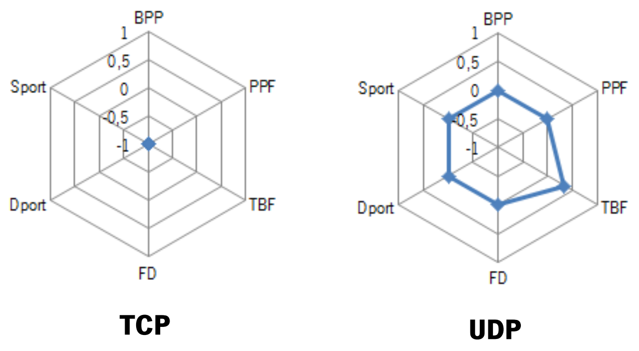


Figure 10: Scanner Communication Fingerprints

The next example refers to a much stealthier behavior. According to the analysis made to the communications between the two hosts, possibly some system fingerprinting occurred. BPP, PPF and TBF values revealed low diversity and it can be observed that the originating host was acting like a server since its Source Ports diversity had values very near zero.

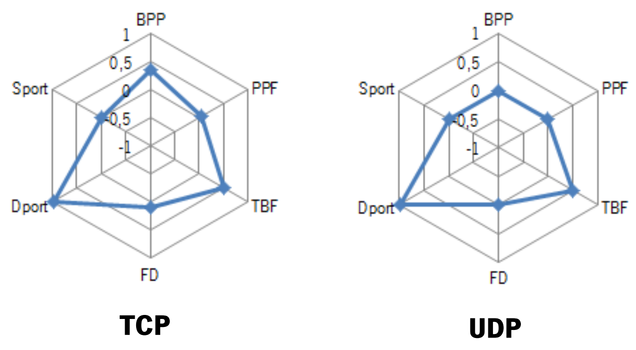


Figure 11: Malicious Host Fingerprints

Not all communications reveal malicious patterns. Below, normal fingerprints from a University of Minho HTTP server are presented. No UDP communications were recorded.

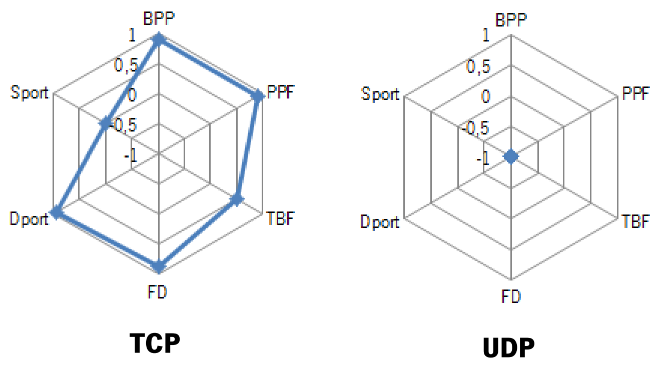


Figure 12: HTTP Server Fingerprints

5.5 Detection Framework Implementation

Based on the analyzed heuristics and shown fingerprinting model, a botnet detection approach framework is here presented.

The proposed detection framework operation is based on five main step-stones:

1. Diversity Index calculation for the proposed heuristics by Source and Destination IP for each twenty-minute time windows using Simpson's Diversity Index formulas/equations presented before.
2. Anomaly Score and Fingerprint computations evaluated for each Source and Destination IP using the different heuristics' diversity indexes. Unidirectional Flows detection is also made in this step. All calculus is done for each of the twenty-minute time windows mentioned in the previous step.
3. Identification of communications with high (above a predefined threshold) TCP or UDP Communication Anomaly Scores (CAS). Hosts showing high number of anomalous connections or unidirectional flows are identified in this step.
4. Second-pass analysis performed on all data collected and gathered in sixty minutes time windows (one hour). In this step, clusters of similar communications (horizontal analysis) are created and recorded, filtering out hosts uniquely showing scan behaviors.
5. Malicious hosts detection and identification based on the second-pass analysis performed in the previous step. Groups of hosts revealing similar connection fingerprints as well as inter-hosts relations are possible to obtain at this phase.

To better comprehend the proposed framework, all step-stones and corresponding anomaly scores calculations, communications clustering and implemented detection control mechanisms are explained in the following pages.

5.5.1 Diversity Indexes Calculation

The Diversity Index Calculation is one of the most important steps in the process since it gathers the data needed for the rest of the proposed detection process. This phase of the detection is implemented in Perl [53] as a nfsen [52] plugin in order to allow fast processing. Diversity indexes are evaluated for the communications in twenty minutes time windows.

A manageable white list implemented in the diversity indexes evaluation phase allows excluding specific hosts from the detection process.

5.5.2 Connection Anomaly Score (CAS) Calculation

After diversity indexes are evaluated, Anomaly Scores are calculated and a corresponding CAS is determined for each Source and Destination host. Total number of Bytes, Packets and Flows exchanged are also recorded in the database. CAS evaluation is based on a weighted average of calculated diversity indexes rounded to the thousandth and multiplied by 1000. This normalizes CAS values in an integer scale that ranges from 0 (not anomalous) to 1000 (maximum anomaly).

Each heuristic presented tries to capture determined characteristics of hosts in a network. It is possible to find high "maliciousness" scores in one heuristic but low on others. In order to truly capture the essence of network normal behaviors and, at the same time, control false positive rates, an inter-heuristic correlation was implemented and tested. The proposed framework uses a combination of four Anomaly Scores (AS_{TBF}, AS_{FD}, AS_{PPF} and AS_{BPP}) each one corresponding to a specific heuristic analysis in order to determine final CAS values. This way, inter-heuristic correlation is also achieved.

In the following equations, Anomaly Score is identified by the acronym AS and Inverse Simpson's Diversity by $\widetilde{D}$.

$$AS_{BPP} = (1 - \widetilde{D}_{BPP}) \times 1000$$

Equation 3: Bytes per Packet Anomaly Score

$$AS_{PPF} = (1 - \widetilde{D}_{PPF}) \times 1000$$

Equation 4: Packets per Flow Anomaly Score

$$AS_{TBF} = (1 - \widetilde{D}_{TBF}) \times 1000$$

Equation 5: Time Between Flows Anomaly Score

$$AS_{FD} = (1 - \widetilde{D}_{FD}) \times 1000$$

Equation 6: Flow Duration Anomaly Score

The Anomaly Scores presented, invert the calculated diversity indexes in order to reflect the true meaning of diversity in the corresponding anomaly score evaluation. High diversity in the Time between Flows heuristic ($\widetilde{D}_{TBF}$) should be considered normal. That way, the corresponding Anomaly Score (AS_{TBF}) should be lower. Since diversity values always range from zero to one, value inversion is achieved by subtracting the respective diversity index to its maximum value (one).

Connection Anomaly Score (CAS) evaluation uses weight definitions applied to each base AS value allowing an administrator to fine-tune the detection framework. Weights are defined in terms of percentages with a total sum of 100%.

Weights	**Description**
$\boldsymbol{W_{BPP}}$	The BPP Weight defines the relative importance of the BPP heuristic and the AS_{BPP} in the Connections Anomalous Score calculation
$\boldsymbol{W_{PPF}}$	The PPF Weight defines the relative importance of the PPF heuristic and the AS_{PPF} in the Connections Anomalous Score calculation
$\boldsymbol{W_{TBF}}$	The TBF Weight defines the relative importance of the TBF heuristic and the AS_{TBF} in the Connections Anomalous Score calculation
$\boldsymbol{W_{FD}}$	The FD Weight defines the relative importance of the FD heuristic and the AS_{FD} in the Connections Anomalous Score calculation

Table 5: Connection Anomaly Score Weights

In the end, for each protocol (TCP and UDP), CAS is determined using the following equation:

$$
\begin{aligned}
&CAS\ (SrcIP, DstIP) \\
&\qquad = AS_{BPP}\ (SrcIP, DstIP) \times W_{BPP} + AS_{PPF}(SrcIP, DstIP) \times W_{PPF} \\
&\qquad + AS_{TBF}(SrcIP, DstIP) \times W_{TBF} + AS_{FD}(SrcIP, DstIP) \times W_{FD}
\end{aligned}
$$

Equation 7: Connection Anomaly Score

CAS values are determined separately for TCP and UDP connections similarly to what is done in the communications fingerprint process. The final CAS value is the maximum value of both TCP and UDP CAS.

5.5.3 First-Pass Detection

The first-pass detection process is based on fingerprint analysis. The *Communication Anomaly Score Threshold (CAST)* is applied to TCP and UDP fingerprints in order to determine which communications should be considered anomalous when evaluating Connection Anomaly Scores (CAS). Based in the assumption of small data exchange in bot operation [39], this framework uses both the *Bytes Per Packet* and *Maximum Bytes Exchanged Thresholds* to control the framework operation range filtering out uninteresting communications for the detection process. At the end of this phase, hosts showing any kind of anomalous behavior are registered (with CAS values above defined CAST).

5.5.4 Horizontal Fingerprint Clustering (HFC) and Second-Pass Detection

The final goal of the detection process is not only to detect all malicious network operations but to find bot operations in all of the network traffic observed. For that matter, and using each host's communications fingerprints, similar malicious interconnected host's activities are clustered. This way, potential botnets can be revealed and False Positive Rates hopefully reduced.

It is important at this moment to point the fact that HFC is performed only to communications involving the anomalous hosts detected in the first pass detection (point 1, 2 and 3 of the defined detection framework's five step-stones).

There are several existing methods to cluster data. Since the goal of HFC is to cluster very similar fingerprints leaving other not so similar outside the clusters, centroid-based clustering methods such as K-Means are not suitable in this context since centroid-based methods have the goal to gather all records in one or other cluster which is not what is intended.

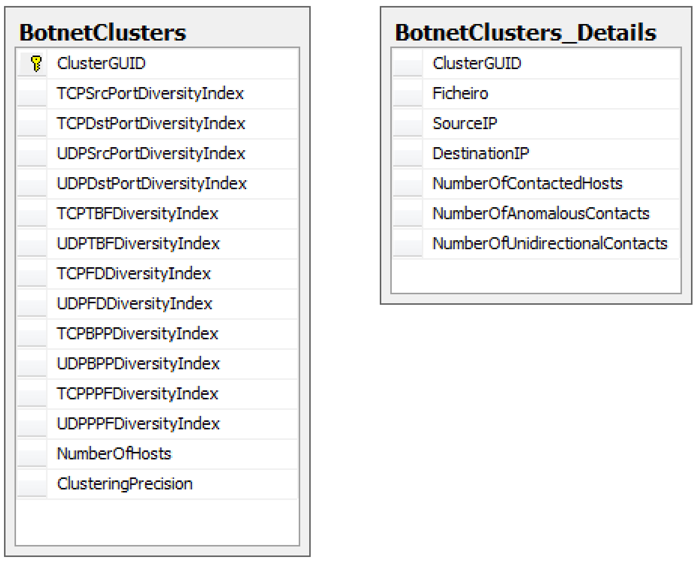
In order to determine similarity between fingerprints, centesimal steps were established in each heuristic's diversity ranges thus predefining possible clusters. A communication is said to belong to a determined cluster when all its heuristic's diversity indexes rounded to the hundredth below are equal to the cluster's heuristics characteristics.

The clustering centesimal value can be changed in the framework and is defined as the *Clustering Precision Threshold (CPT)*. A centesimal (0.01) *Clustering Precision Threshold* guarantees that all communications in a cluster have fingerprints with each characteristic's value (BPP, TBF, etc.) distancing no more than 0.009(9) from each other.

To control the Horizontal Analysis (clustering) sensitivity, a new threshold was needed. The *Horizontal Fingerprint Analysis Threshold (HFAT)* allows defining the minimum number of hosts that a cluster must have in order to be considered in the detection process.

All possible clusters values are first determined and inserted into a database table (BotnetClusters) in every hour time-window. GUIDs (Globally Unique Identifiers) are assigned to each possible cluster at this time defining clusters identifiers. After clusters are predetermined for the hour, all communications involving anomalous hosts are then clustered and inserted in the cluster details database table (BotnetClusters_Details).

The database tables supporting the clustering process are shown in Figure 13.



BotnetClusters

- ClusterGUID
- TCPSrcPortDiversityIndex
- TCPDstPortDiversityIndex
- UDPSrcPortDiversityIndex
- UDPDstPortDiversityIndex
- TCPTBFDiversityIndex
- UDPTBFDiversityIndex
- TCPFDDiversityIndex
- UDPFDDiversityIndex
- TCPBPPDiversityIndex
- UDPBPPDiversityIndex
- TCPPPFDiversityIndex
- UDPPPFDiversityIndex
- NumberOfHosts
- ClusteringPrecision

BotnetClusters_Details

- ClusterGUID
- Ficheiro
- SourceIP
- DestinationIP
- NumberOfContactedHosts
- NumberOfAnomalousContacts
- NumberOfUnidirectionalContacts

Figure 13: Clustering Database Tables

The final phase of this step begins by calculating an Anomaly Score for each Source IP or Host. Host Anomaly Score (HAS) is calculated using the equation below.

$$HAS\ (SrcIP) = \frac{Number\ of\ Anomalous\ Contacts\ (SrcIP)}{Total\ of\ Contacts\ (SrcIP)} \times 1000$$

Equation 8: Host's Anomaly Score

Similar to the Connection Anomaly Score Threshold used to control de First-Pass detection phase, the Second-Pass phase is controlled by the Host Anomaly Score Threshold (HAST) and Minimum Number of Anomalous Contacts Threshold (MNACT). MNACT influence detection by imposing a lower limit to the number of anomalous connections a host starts in order to be considered for detection. This threshold allows systems administrators discard fortuitous anomalous contacts detected.

Beyond HAS, a Unidirectional Anomaly Score (UAS) is also evaluated to help identifying scan behaviors.

$$UAS\ (SrcIP) = \frac{Number\ of\ Unidirectional\ Contacts\ (SrcIP)}{Total\ of\ Contacts\ (SrcIP)} \times 1000$$

Equation 9: Unidirectional Anomaly Score

At this point, and to control scan behaviors detection, two thresholds are used: the *Unidirectional Anomaly Score Threshold (UAST)* and *Minimum Number of Unidirectional Contacts Threshold (MNUCT)*. The percentage of unidirectional contacts performed by a single host is compared to the UAST. All hosts with a number of unidirectional contacts above the defined MNUCT and a UAS above the defined UAST is then considered to be a *Scanner*.

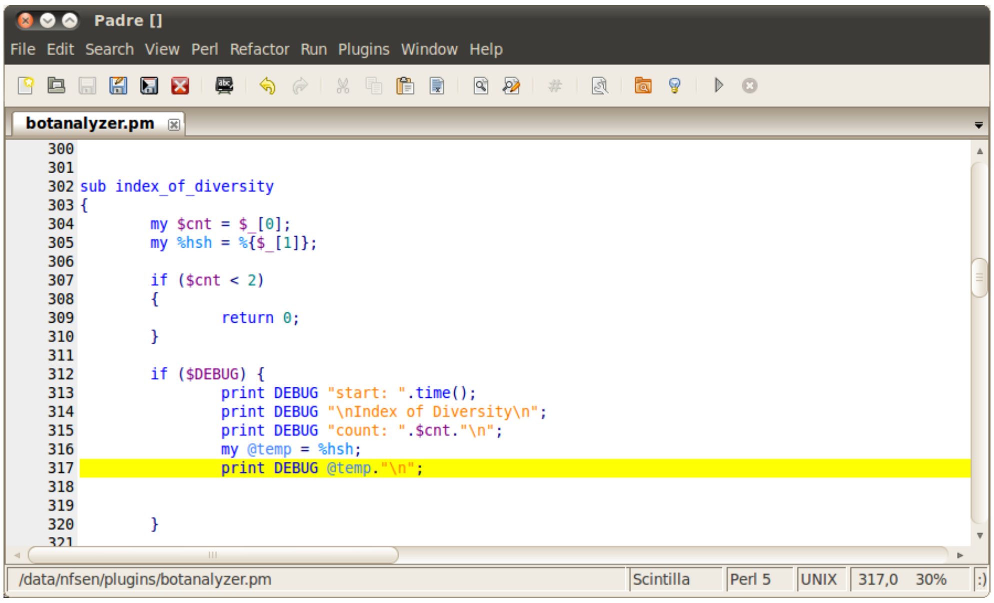
5.5.5 Alerts

The final step of the process is to record all hosts identified as anomalous in order to deliver an alert to system administrators. Final detection step can be performed with or without clustering analysis. In order to use this detection framework for botnet detection clustering analysis must be used.

5.6 Prototype Implementation

The Nfsen plugin development was made using the open source Perl [53] Padre v0.45 [56] IDE[8] installed in the Ubuntu virtual environment later used for testing.

[8] Integrated Development Environment


Padre []
File Edit Search View Perl Refactor Run Plugins Window Help
botanalyzer.pm
/data/nfsen/plugins/botanalyzer.pm
Scintilla
Perl 5
UNIX
317,0
30%

```


sub index_of_diversity
{
        my $cnt = $_[0];
        my %hsh = %{$_[1]};

        if ($cnt < 2)
        {
                return 0;
        }

        if ($DEBUG) {
                print DEBUG "start: ".time();
                print DEBUG "\nIndex of Diversity\n";
                print DEBUG "count: ".$cnt."\n";
                my @temp = %hsh;
                print DEBUG @temp."\n";


        }
```

Figure 14: Padre Integrated Development Environment

The NFSight plugin [57] was clearly an inspiration for the prototype here presented and along with the Nfsen plugin development documentation was the basis for the BotAnalyzer plugin development.

All diversity indexes evaluated by the nfsen plugin are saved to raw text files and imported to a Microsoft SQL Server Database through automated procedures in order to implement the last phases of the detection such as the clustering analysis.

```
EXEC FillHostsAnomalyScore '201111161100', '201111161200'

DELETE FROM Alerts
EXEC dbo.Detect 700
EXEC EvaluateAccuracy_v2

-----------------------------------------------------------------
-- Clustering
-----------------------------------------------------------------
EXEC Cluster
EXEC Cluster_Details 10, 950

-----------------------------------------------------------------
-- Hourly Alerts
-----------------------------------------------------------------
DECLARE @UseCLAS as bit
SET @UseCLAS = 1
DELETE FROM HourlyAlerts
EXEC HourlyDetection '201111161100', 10, 990, 1, 1, @UseCLAS
EXEC EvaluateHourlyAccuracy
```

Figure 15: SQL Server Scripts Test Examples

5.7 Conclusions

Several difficulties were faced during the proposed prototype implementation. The hardest constraints to overcome were found in available operating system memory, storage and overall detection framework performance since diversity indexes calculation methods involved the use of large arrays of data in memory. The use of the PERL programing language and the nfdump tools were fundamental to minimize such limitations in the development environment.

It was also possible to observe that the time window used when evaluating the diversity indexes was an important aspect to consider. Larger time windows guaranteed more precise diversity indexes. That way, the use of a twenty minutes time window used in the detection framework was solely motivated by the development environment memory constraints (2 Gb). Tests made with larger time-windows on daily periods with less traffic produced more precise Anomaly Score values.

Since this framework's goal is to be a proof-of-concept and not a fully functional product, some implementation aspects were consciously overlooked in order to focus development in detection algorithms and strategies. In this context it is clear that, although being automated, the import of diversity indexes files to SQL Server could be eliminated from the process by making the developed nfsen plugin write calculated diversity indexes directly to the database.

6. Experimental Setup, Results and Analysis

"Try and fail, but don't fail to try." Stephen Kaggwa

In this section it is first described all the research environment and test procedures used to validate the proposed prototype. Variations on the prototype's control variables used (thresholds) are then analyzed in order to evaluate detection accuracy and determine in what way threshold definitions influence the system's detection rates. Above all, it was important to understand and feel different ways of testing a detection framework such as this one.

6.1 Test Datasets Characterization

All traffic datasets used in detection prototype development and tests were collected by the Communications Services of University of Minho in the university's edge network equipment. These datasets were all gathered using ntop [47] and nfdump [48] tools and included all the university network traffic leaving and entering both the Braga and Guimarães campi. Network traffic used in the prototype tests and development included collected netflows from the 11th and 12th January 2011 as well as from the 16th November 2011 for the ultimate test runs.

Dataset Date	Description	Dataset Size
11 Jan 2011	289 nfcapd files. Each file archives five minutes of inbound and outbound traffic from the University of Minho campi. Netflows collected on Tuesday, 11th January 2011.	6,4 Gb
12 Jan 2011	289 nfcapd files. Each file archives five minutes of inbound and outbound traffic from the University of Minho campi. Netflows collected on Wednesday, 12th January 2011.	7,6 Gb
16 Nov 2011	289 nfcapd files. Each file archives five minutes of inbound and outbound traffic from the University of Minho campi. Netflows collected on Wednesday, 16th November 2011.	6,22 Gb

Table 6: Test Datasets – Description and Size

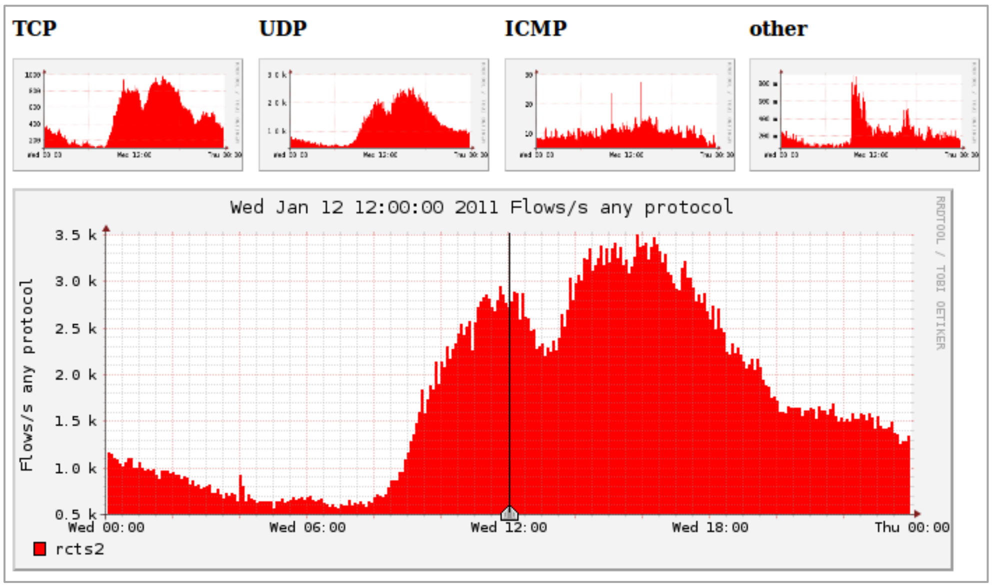


Figure 16: Typical University of Minho 24-hour traffic shown in Nfsen (12th January 2011)

Flows used for prototype development and testing were built from traffic collected in a Catalyst 4500 L3 Switch through a mirrored port. Flows where created from captured traffic using nprobe [47] installed on a Linux CentOS 5.3 box on top of a Xeon 5130@2.0GHz (Dual) processor with 4Gb RAM memory.

After netflow creation, the nfcapd [48] daemon installed in a Linux Fedora Core 5 running on top of a Xeon E5310@1.60GHz (Quad) processor with 4Gb available RAM provided netflow capture and storage.

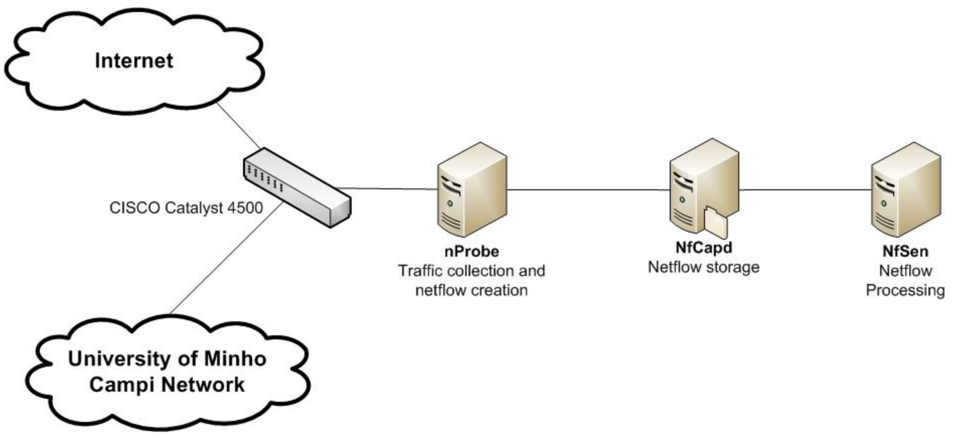


Figure 17: Netflow Capture System

To test the framework's sensitivity, several Botnet Blacklists were retrieved from well-known open projects in the security community. IPs contained in these blacklists overlapped as it would be expectable. After removing all duplicate IPs, the final Malicious Hosts database table contents was distributed according to the table shown below.

Source	Number of IPs
sucuri.net [58]	3256
malc0de.com [59]	2783
dshield.org [60]	2600286
abuse.ch [61]	329
spam-ip.com [62]	44017

Table 7: Botnet Blacklists Sources and Number of Hosts

The selection of the above blacklists took in consideration that the malicious hosts reported should be representative of bot operations containing not only spammers or scanners, for instance. Selected blacklists also included bots performing different types of attacks: Distributed Denial of Service (DDoS), SQL Injection, PHP attacks, etc. Another important aspect is that selected blacklists only included bot detections in the same time-window of the University of Minho datasets under test.

Independently of any malicious activity recorded, hosts registered in botnets blacklists were found in the analyzed University of Minho datasets. Two charts characterizing the presence of such hosts in the 16th November 2011 dataset can be found below.

The first chart (Figure 18) represents the number of distinct hosts found in the analyzed dataset on each hourly period. The second chart represents de number of distinct hosts that were registered in the blacklists used for the same periods. It was also possible to detect on such blacklists the existence of University of Minho hosts.

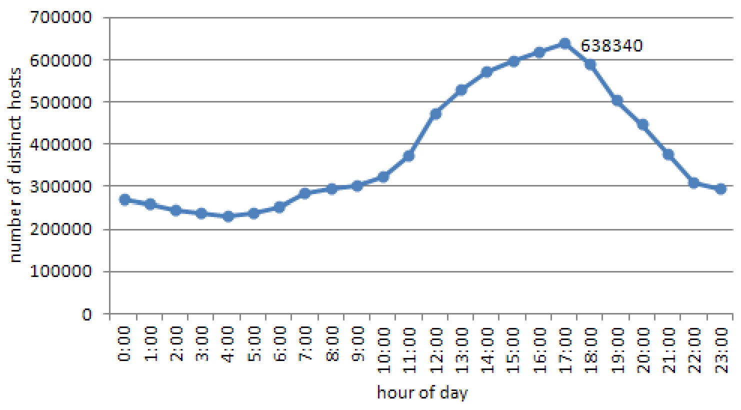


Figure 18: Distinct active hosts for each hourly period (16th November 2011)

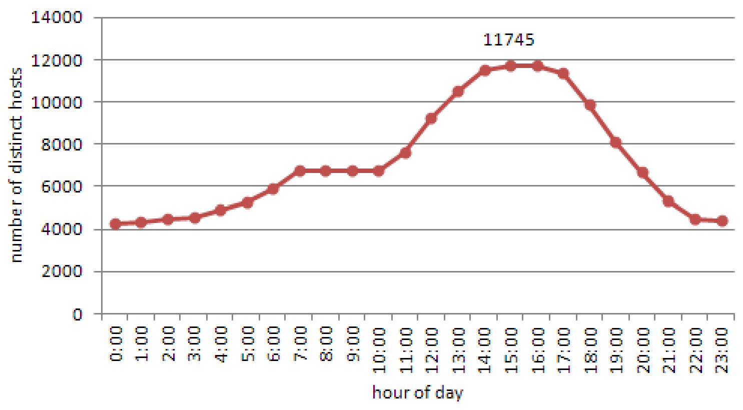


Figure 19: Distinct active hosts for each hourly period present in blacklists (16th November 2011)

Values presented in the charts above don't include hosts involved in connections to and from whitelisted hosts such as the University of Minho HTTP and SOCKS proxies.

6.2 Prototype Test Environment

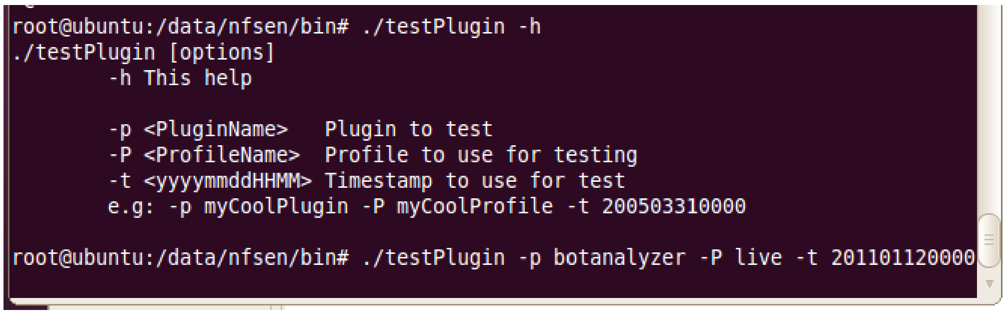
After netflow collection, flows were copied to an nfsen [52] repository. Nfsen 1.3.5 was installed in an Ubuntu 10.04 Virtual Machine and configured to include the test netflow collection in its available datasets.

Virtual Machine infrastructure was provided by the VMWare Server v2.0.2 software [63] running on a P8600 Core2 Duo 2.40GHz Intel Processor with 4 Gb RAM memory space available. Windows 7 64 bits Professional Edition was the host operating system used.

The virtual machine supporting the test environment was created having one single virtual processor, 2 Gb RAM, 40 Gb disk space as well as bridged network connections.

The developed nfsen plugin (*BotAnalyzer*) was configured and activated in the environment described. In order to test the detection process on the available datasets, the nfsen *testplugin* tool, included in nfsen installation, was used. This tool simulates real nfsen operation.

To better understand the use of the *testplugin* tool, a command line execution of the B*otAnalyzer* plugin is shown below.



```
root@ubuntu:/data/nfsen/bin# ./testPlugin -h
./testPlugin [options]
        -h This help

        -p <PluginName>   Plugin to test
        -P <ProfileName>  Profile to use for testing
        -t <yyyymmddHHMM> Timestamp to use for test
        e.g: -p myCoolPlugin -P myCoolProfile -t 200503310000

root@ubuntu:/data/nfsen/bin# ./testPlugin -p botanalyzer -P live -t 201101120000
```

Figure 20: nfsen "testplugin" command-line tool

In the command above, the system is instructed to run the *BotAnalyzer* plugin on the netflows collected. *BotAnalyzer* then processes the network traffic collected in the first twenty minutes of the 12th January 2011. Analyzed netflows were stored in the “live” profile dataset of the nfsen netflow repository.

6.3 Testing Methodology

In this section it is explained all the important testing decisions and methods that influenced the final results presented.

Sensitivity evaluation of a system such as the proposed one is not a simple task since tight control on the analyzed data would be necessary to precisely determine type I (false positives) and type II (false negatives) statistical errors. On the other hand, the use of controlled datasets can lead to biased test results locked to the datasets used. Tests made using real datasets and information are prone to reveal, unwillingly, some aspects of the framework under test and of traffic patterns that could be hidden somehow. With this in mind, the goal was to sense the detection framework sensitivity to threshold changes and its capacity to approach the ideal low false positive scenario.

To determine the sensitivity and specificity of the detection Framework, True Positive and Negative Rates (TPR and TNR), as well as False Positive and Negative Rates (FPR and FNR respectively) were determined using the 24 hours' time window (one day) dataset from the 16th of November 2011.

For the evaluation of TPR, TNR, FPR and FNR, several base values were defined and determined for the time windows under analysis:

1. Number of hosts (H)
2. Number of Malicious Hosts (MH)
3. Number of Detected Hosts (DH)
4. Number of Detected Malicious Hosts (DMH)
5. Number of Undetected Normal Hosts (UNH)

MH values were evaluated by counting the number of hosts present in the sample that coexisted in the botnets blacklists or undoubtedly presented scanning behaviors. DH values were determined by simply counting the number of detections made by the framework. Finally, DMH values were reached by determining the number of detected hosts that were present in the botnets blacklists or were undoubtedly scanners. The UNH values were evaluated by counting all the supposedly normal hosts present in the sample and that were not detected.

Based on the values above, True Positive (TP), True Negatives (TN), False Positive (FP) and False Negative (FN) values were determined using evaluation methods capable of avoiding quantification errors such as *base rate fallacies* [18], [64].

True Positives and True Negatives were evaluated directly by using the DMH and UNH values respectively.

$$TP = DMH \qquad\qquad TN = UNH$$

Equation 10: True Positives and True Negatives

False Positives were defined by the difference between the total detected hosts and detected hosts that belonged to the malicious IPs dataset.

$$FP = DH - DMH$$

Equation 11: False Positives

False Negatives could then be defined by the difference between all proven malicious hosts in the sample and the corresponding number of detected ones.

$$FN = MH - DMH$$

Equation 12: False Negatives

The Sensitivity or True Positive Rate (TPR) was defined by

$$TPR = \frac{TP}{TP + FN}$$

Equation 13: True Positive Rate

The Specificity or True Negative Rate (TNR) was defined by

$$TNR = \frac{TN}{TN + FP}$$

Equation 14: True Negative Rate

False Positive Rate (FPR) was defined by

$$FPR = \frac{FP}{FP + TN}$$

Equation 15: False Positive Rate

False Negative Rate (FNR) was defined by

$$FNR = \frac{FN}{FN + TP}$$

Equation 16: False Negative Rate

In the last phase of detection, sensitivity analysis was extended in order to consider hosts marked as scanners to be true positives. This decision was based on the fact that scanning behaviors were very clear to observe and represented without question malicious activity.

In order to sense the proposed framework threshold values were varied to determine the system's sensitivity and specificity.

In a system's administrator viewpoint, it is not so important that some thousands of malicious hosts and corresponding activities are not detected. It is crucial, though, that all the detections made are true and that the team in charge of network administration is able to check and treat them all. With this in mind, a decision was made to also evaluate de rate of true and false detections over the overall detections.

The evaluation of true and false positives based on the selected botnet blacklists is like walking on quicksand. There's no guarantee that the blacklisted hosts were performing malicious activities in the time windows analyzed and the same can be said about hosts not blacklisted that could be performing malicious activities. So, why use them?

The main goal was to drastically reduce the number of possible malicious hosts that needed confirmation analysis. Since no access to packet data was possible, malicious activity confirmation could rely only on the flow data available. Observing recorded flows it is simple to detect some abnormal behaviors such as scanning. Other behaviors lie in the not so thin limbo between normality and abnormality. Crossing flow observation with external information of hosts with bot behavior (blacklists) allowed confirming much more confidently the detections made.

6.4 Detection Sensitivity and Specificity Analysis

As seen in the dataset characterization section, more than 4000 and up to 11000 blacklisted hosts can be observed in the traffic recorded. Analysis of such flows is a highly time consuming task and there was no option left but to leave that analysis to posterior studies in order to keep the initial thesis planning from slipping. Having this in mind, some tests were performed on the 16th November 2011 dataset in order to evaluate the framework sensitivity and specificity to threshold changes.

In this thesis context, *Sensitivity* corresponds to True Positive Rate or the ability to correctly detect malicious hosts. *Specificity*, on the other hand, relates to True Negative Rates or the ability to tell malicious hosts from those who are not.

First-pass detection revealed to have low sensitivity to Connection Anomaly Score Threshold (CAST) variations. Changes on the weights used to evaluate the Connection Anomaly Score (CAS) of each Source and Destination IP pair, produced small changes on final detection rates. In order to verify the framework's response to CAST variations some first-pass tests were executed on the first hour of network traffic from 16th November 2011. Such tests' results are presented below.

Weights used to calculate the CAS values used in the tested first-pass detection process were:

$$W_{BPP} = 30\%, W_{PPF} = 30\%, W_{TBF} = 20\% \; and \; W_{FD} = 20\%$$

The sensitivity of the framework was tested on CAST values varying from 500 (lower connections anomaly scores) to 1000 (highest connections anomaly scores). A table with used CAST values and corresponding detection rates is presented below. A change in result values is clearly seen between CAST values 750 and 800. For a True Positive Rate (TPR) drop of about 9%, there is a huge drop on False Positive Rate (FPR): about 90%.

CAST	TPR	FNR	FPR	TNR
500	99,7637	0,236269	98,2111	1,78888
550	99,7281	0,271871	97,9671	2,03291
600	99,5242	0,475774	96,7603	3,2397
650	99,4951	0,504903	96,5342	3,46577
700	99,4271	0,572871	96,0249	3,97515
750	99,343	0,657022	95,3245	4,67549
800	90,2644	9,73557	13,4694	86,5306
850	89,7984	10,2016	9,63345	90,3666
900	89,6818	10,3182	8,56748	91,4325
950	89,4909	10,5091	7,08949	92,9105
1000	89,4909	10,5091	7,07988	92,9201

Table 8: CAST variation framwork's response (rates in %)

Shown detection rates were evaluated comparing detected hosts against registered blacklisted IPs and proved scanning behaviors.

Results are more perceptible on the *Receiver Operation Characteristic (ROC) Plot* and *Sensitivity vs Specificity* charts presented below.

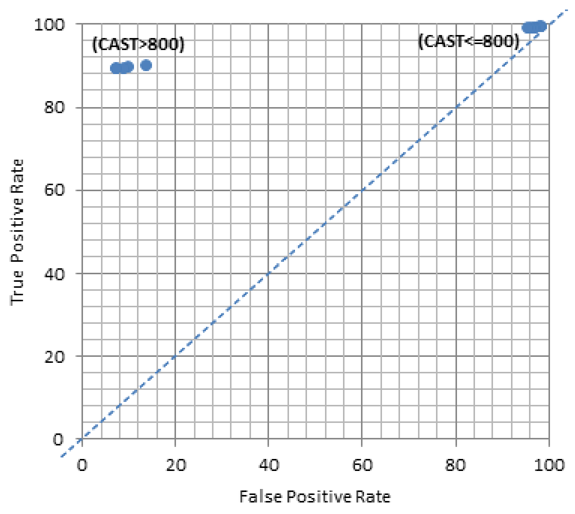


Figure 21: ROC Plot for CAST variation

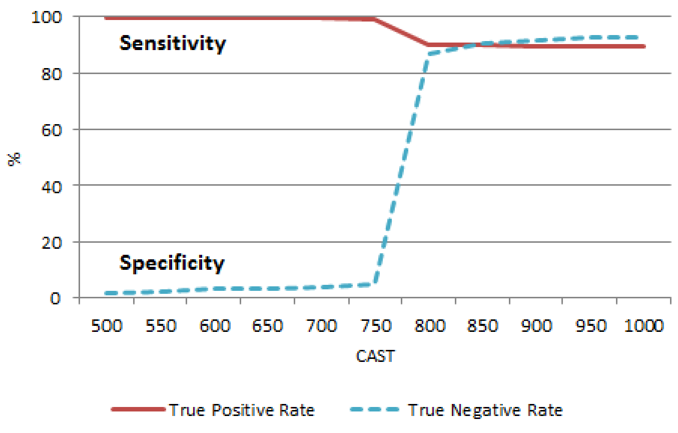


Figure 22: First-Pass Detection - Sensitivity versus Specificity

The charts above reinforce what was stated before clearly showing a cut-off CAST value around 800. Below this value, True Positive Rate or *Sensitivity* prevails over True Negative Rate or *Specificity*. Above that value *Specificity* prevails over *Sensitivity*.

Specificity and *Sensitivity* alone are not used to make any sort of framework's evaluation to avoid some of the pitfalls mentioned in [18]. To completely evaluate a detection system such as this, the cost of not detecting true anomalies (false negatives) must also be evaluated. This cost can be higher in some organizations and lower in others.

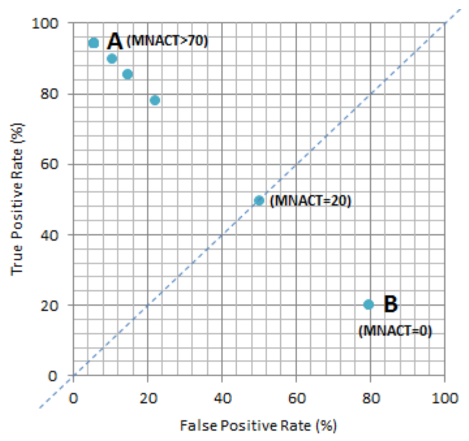
As stated in [18] and [64], base rate fallacy can reveal high number of alarms in low relative false positive rates. Even with a low false positive rate, really high alarm rates are hard or impossible to handle by system administrators. Low false positive rates don't imply a low number of false positives when the alarm rate is very high. Since alarm handling capability is primordial in a detection system, *Specificity* should prevail over *Sensitivity*. That way, CAST values above 800 were used for first-pass detections in the rest of the tests made lowering base rate detection and reducing the number of hosts and connections handled to the second-pass detection phase.

Second-Pass detection is the final detection step happening just before the alert phase. Results of the tests made on the global response of the framework are presented below. Detection response was tested with and without clustering analysis.

The framework's sensitivity to changes on the Minimum Number of Anomalous Connections Threshold (MNACT) per host was tested at this time. Detection rates evaluated were again based on the blacklists collected and already mentioned in this section. A Host Anomaly Score Threshold value of 950 was used both for the clustering and second-pass detection phases. The selected value intended to reduce the number of alarms.

Although Receiver Operating Characteristics (ROC) curves should not be used alone to determine optimal operation points for Intrusion Detection Systems [65], a very simple (with few data points evaluated) ROC plot for the first-pass detection was created. Test goals were not to fine tune the detection system itself but to test its sensitivity and reaction to MNACT and HAST variations.

It could be verified that Clustering Analysis smoothed the variations in system's detection response to MNACT changes as can be seen in the ROC plots of Figure 23.



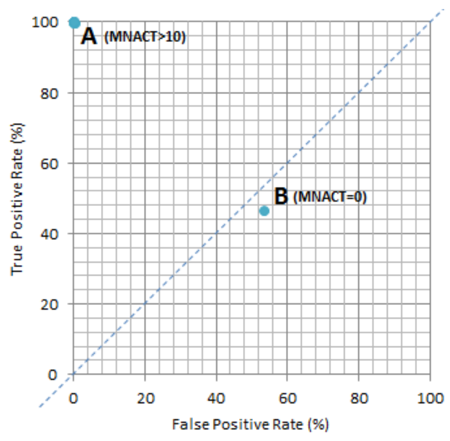


Figure 23: MNACT ROC plots with (left) and without (right) clustering analysis

All data used for the ROC plot charts presented above was extracted from the first hour of traffic of the 16th November 2011. A Host Anomaly Score Threshold value of 950 was used both for the clustering and second-pass detection phases tests. In order to create the plots above, the framework was submitted to 21 tests with MNACT varying from 0 to 200 in 10 unit steps. In the plots above and when clustering analysis is used, it can be seen that to all MNACT values above 70 the framework's detection response is very similar.

It can also be seen that the detection results using clustering analysis were worse than without clustering analysis. Such results can be explained by the simple fact that detection using clustering analysis introduces the variable of inter-host anomalous connections being penalized by the selected evaluation method.

Unidirectional Anomaly Scores (UAS) heuristic is very important in the proposed detection framework since it undoubtedly indicates anomalous (and potentially malicious) behaviors. With the introduction of clustering analysis, many considered malicious hosts with scan-only behaviors are not detected since they must have bi-directional flows in order to participate in a potential botnet cluster.

HAST changes sensitivity and the use of clustering analysis were then evaluated on the full 16th November 2011 dataset against the selected blacklists and proved scanning behaviors. A MNACT value of 70 was used in the HAST changes sensitivity according to the results of the MNACT variation tests executed. Charts below analyze the response of the framework throughout the day of 16th November 2011.

Fixing a HAST of 950 and a MNACT of 70, 24 hourly periods were submitted to the framework's detection system. Such test results are presented in the following pages beginning by analyzing detections not using clustering analysis.

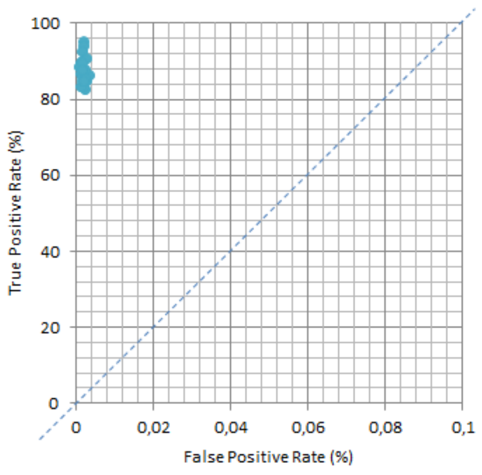


Figure 24: ROC Plot for detection without clustering analysis for MNACT=70 and HAST=950

In the shown ROC Plot it can be seen that TPR values vary between 80% and 95% for all the 24 hours of the day while there is a small variation in FPR (values around 0,001%). On a network administrator point of view, values like the ones presented can be deceiving, especially if no clustering analysis is in place. On the dataset under analysis, every hour, a minimum of 22000 distinct hosts performed network scans on the network. Analyzing and responding to such behaviors is a lifetime task that no network administrator would like to take.

The TPR/FPR points presented in the ROC Plot chart of Figure 24 correspond to the true and false number of detections presented below.

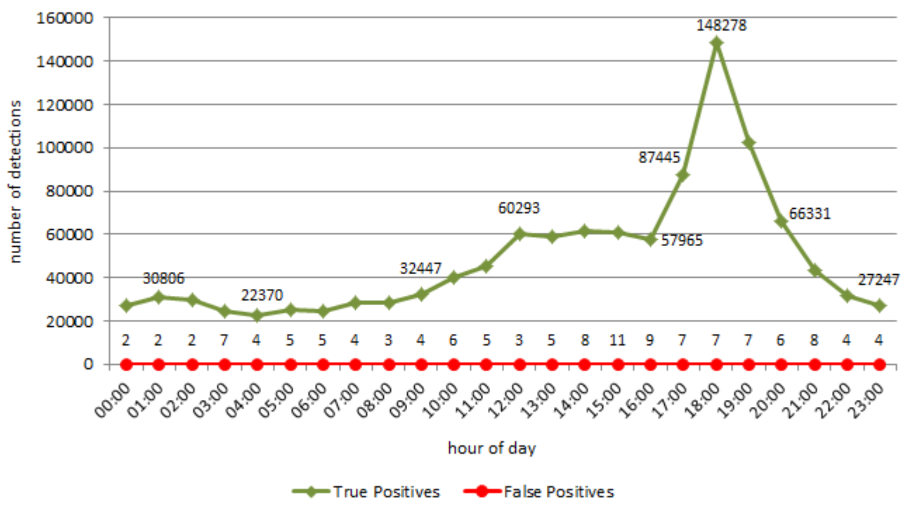


Figure 25: Hosts detected without clustering analysis for MNACT=70 and HAST=950

Such a number of detections can be overwhelming for a network administrator. In order to lower the number of detections, more restrictive threshold values could be applied. As seen before, clustering analysis can be used to filter out pure scanners, grouping at the same time anomalous hosts with similar behaviors. Similarly, other detection orientations could be implemented to filter out other detectable network anomalous behaviors.

Results of the same previous test but using clustering analysis are shown in Figure 26.

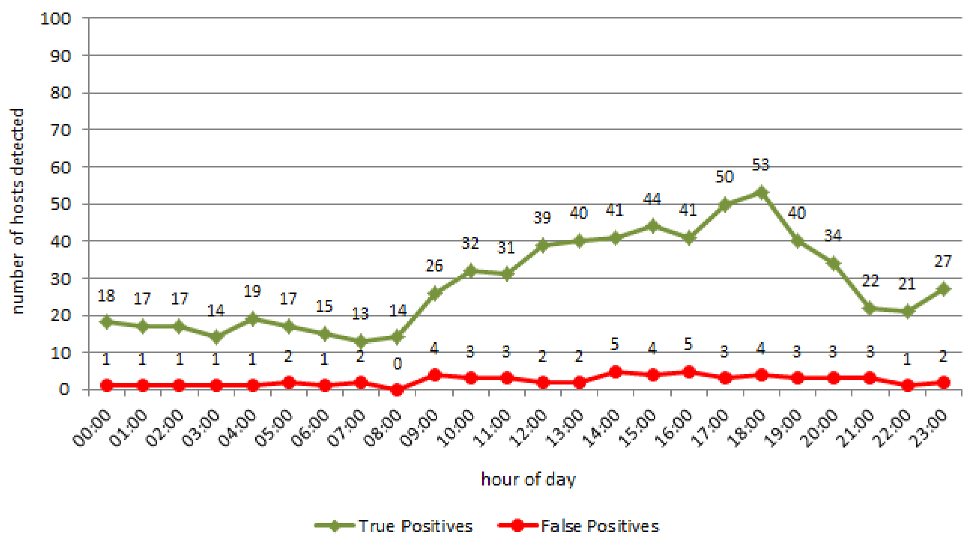


Figure 26: Hosts detected using clustering analysis for MNACT=70 and HAST=950

It's clear from the chart above that clustering analysis decreases the overall detection rates. True Positive Rate largely decreases accompanied by a very small decrease in the False Positive Rate.

Keeping MNACT value unchanged (70), an increase in the Host Anomaly Score Threshold (or HAST) was then experimented in order to test the framework under heavier threshold restrictions.

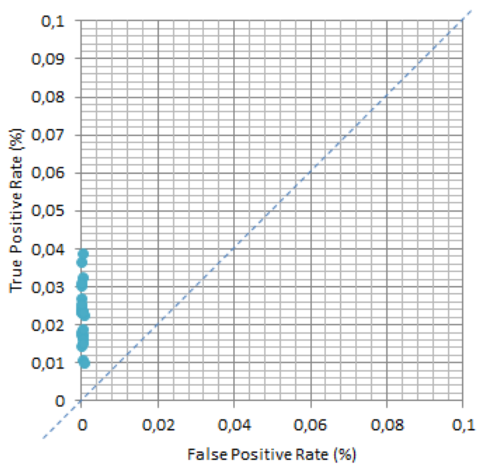


Figure 27: ROC Plot for detection with clustering analysis for MNACT=70 and HAST=990

Although False Positive Rates (FPR) remains sensibly the same, there's a huge drop in the True Positive Rates (TPR) similar to the one already verified with previous tests with clustering analysis. Even with the TPR values shown in the previous ROC Plot under 0,04%, there were some considerable detections made as can be observed in the following chart.

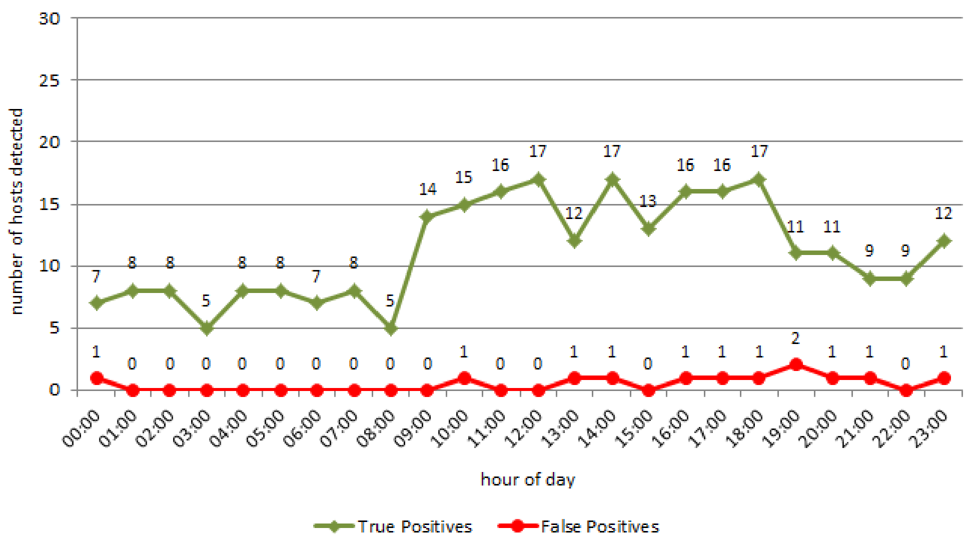


Figure 28: Hosts detected using clustering analysis for MNACT=70 and HAST=990

An important aspect of detection can be deducted from the results above. A low detection rate can sometimes be a good thing providing that low FPR also exists. It can be more productive to a network administrator to focus on few but also highly anomalous true detections then to be flooded with thousands of detections that although representing anomalous behaviors don't correspond to the more dangerous ones (or the ones sought).

6.5 Performance and Storage: Real-Time?

After system's accuracy evaluation was made, an important question could be placed: can such a system be implemented in a real-time scenario?

The answer to such question lies in the time needed to process a full hour of traffic. If the detection framework is able to process all the network hourly traffic collected in less than an hour, than it is feasible to implement this framework in such network.

Performance evaluation was made over the daily University of Minho dataset of the 16th of November 2011. This represented a normal working weekday (Wednesday).

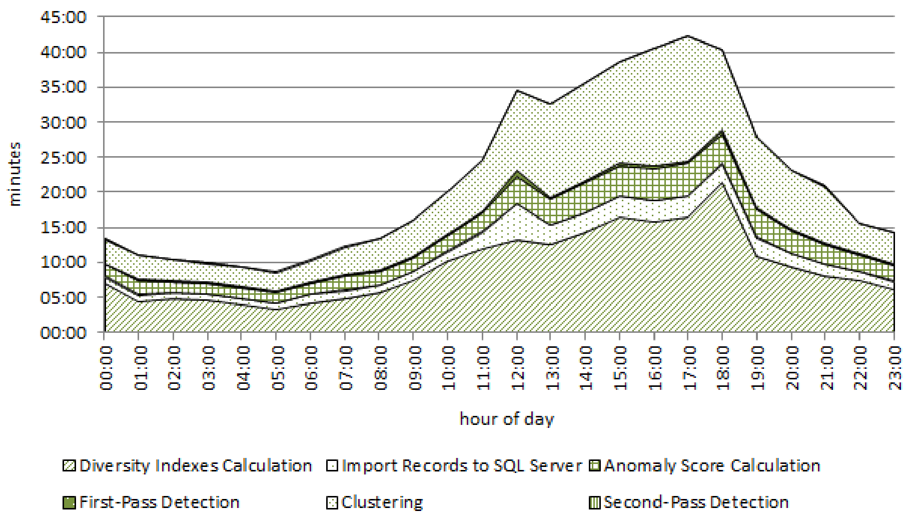


Figure 29: Performance Evaluation Chart

Although all performance constraints found, it is possible to observe from the chart above that in the worst case and for a University of Minho normal day's network traffic, the detection framework took less than 45 minutes to process the traffic collected in an hour and produce corresponding alerts.

It can be also observed that the heavier time consuming processes are the diversity index calculations and clustering. These processes should be included in the first performance aspects to improve.

Performance can also be improved by removing the import records phase. This is a relatively simple task since the perl nfsen plugin developed is able to access the majority of known database systems.

Another aspect is the storage needs of the detection framework. Besides the storage used to keep all the flow files, diversity indexes text files produced by the developed plugin and the SQL Server database are byte consuming processes.

For the day under performance analysis it was needed 2,47 Gb of disk space for the files generated by the Nfsen plugin. These files included all the calculated diversity indexes for the communications between hosts as well as bytes and flows exchanged.

The SQL Server database, after the daily detection run ended up with 9,34 Gb.

6.6 Final Results and Conclusions

The good news was that it was possible to detect anomalous behaviors both in the first and second-pass processes.

It was possible to verify, though, that when there are few connections recorded for each Source IP, the diversity calculations results tend to be biased. This fact ends up contributing to higher false positive rates in some of the firs-pass detection runs. It is then important to extend as much as possible the time window used in diversity indexes calculation.

An important feature not included in the presented model would be the possibility of creating inter-periods relations marking detected malicious hosts in order to use them in posterior detections confirmation.

Beyond all faced difficulties, the memory and CPU constraints ended up playing an important role on the implementation decisions. Although these forced decisions could be avoided with other resources in hand they ended up contributing for a better final implementation of the prototype framework.

Storage needs was not a preoccupation in this research. At every step of the development process there was a need to have all the possible information at hand to continuously validate the implementation and detection process. This need produced many unneeded data records that could be removed from a final framework implementation.

Since the first approach for ECIW'11, better scan detection heuristics were developed. The later use of Unidirectional Flows analysis revealed to be more precise than the initial Bytes per Packet (BPP) similarity used. Service (Port) oriented analysis was experimented initially and later dropped for simplicity. With more CPU and memory available in test systems, the initial Service Oriented approach should be reanalyzed and integrated in the current framework.

Future work could include the analysis of alternate statistical approaches such as Random Forest Trees [66] or algorithms such as the case of streaming algorithms [67].

All false positives detected were related to valid network services running inside the campi network. Among such services there was DNS, DHCP, VPN and other authentication related services such as Netbios or Kerberos. These false positives could be eliminated simply by whitelisting them after verification. Such whitelist records could also have associated time-to-live (TTL) values in order to force new verifications from time to time. The problem of blind-whitelisting network services servers is that such action could be hiding entire botnets; beyond providing the expected service (DNS, DHCP, etc.), servers can also be operating as bots or C&C servers.

It was possible to see that the cut-off value detected on the first pass detection phase was influenced by the fact that unique flows were not valorized by the detection algorithm. This response from the detection framework should be reviewed because many of the botnet related traffic detected included hosts with single bidirectional flows to many hosts.

This fact brought forward the lifetime behavioral changes of a botnet. When not trying to recruit more bots or attacking, botnet network traffic can be reduced to a minimum "keep-alive" phase. To improve detection capabilities in such botnet periods different heuristics must be created. One possibility is to create a *Deep-Vertical-Analysis* by crossing flow data sent by hosts with corresponding flow answers. *Deep-Horizontal-Analysis* could then be used to create a *Wide-Horizontal-Analysis* to detect stealth C&C botnet communications. This detection method should not use scanning detection data since this behavior relates to a different phase of the botnet lifetime.

Since network administrators have constraints such as the human resources and time available to investigate intrusions in the supervised system, a final threshold could be added to the system: the pretended *Number of Detections*.

In order to determine the best *Number of Detections* and other thresholds for the supervised network, the blacklist approach could be used to evaluate sensitivity on past data and define thresholds accordingly.

As seen before, clustering analysis can be used to filter out pure scanners grouping at the same time anomalous hosts with similar behaviors. In this context, existing heuristics can be improved as well as more heuristics can be developed in the future to allow the identification of the type of detected anomalies such as what attack is being executed on the network.

Another possible improvement is the use of IP addresses and domain name crossing to improve the clustering process. Domain fluxing usage by botnets can result in many IPs responding to one single domain. Using parallel domain clustering could improve clustering accuracy.

A different threshold scheme could also be implemented with the definition of *Minimal Thresholds*. Minimal Thresholds would make the framework detect more anomalies allowing the creation of a host reputation system by ranking HAS values.

Finally, a word must be said about the method used to analyze the framework's response. In order to fully evaluate a system like the one proposed many more aspects of detections should have been considered along with distinct detection confirmation methods.

In future work, the use of real known bot traffic injected in analyzed netflows should be considered as well as, if possible, packet inspection to confirm the detections made. The cost evaluation of missed detections is also an important aspect to study in the future. Proper metrics should be defined in order to create the basis of a coherent Intrusion Detection Systems evaluation framework.

7. Discussion, Future Work and Conclusions

"When you get there, there isn't any there there." Gertrude Stein

Gertrude Stein put it well in words: the end of the road isn't really the end but just the beginning of a new one. A step towards a better understanding of the botnet phenomenon was taken and this research was also useful in the sense that allowed the exploration of some tools.

The development of a prototype botnet detection framework was really important since it promoted the contact with real traffic and all the difficulties related to detection algorithm implementation. It was confirmed, though, that it is possible to detect network anomalies by solely analyzing netflow attributes.

Although the heuristics proposed in this thesis lack further investigation and development, they were able to pinpoint some important network events that, when drilled-down, presented real deviations from normal network traffic. The process of framework evaluation to determine if such anomalies represent real, botnet-related, malicious traffic is still an area to explore.

The developed fingerprinting and corresponding visualization method based on the diversity calculations brought a new perspective to the way inter-host connections are seen and perceived.

Besides all future work already mentioned throughout this thesis that could improve the proposed detection process, it would be important to validate the framework using distinct methodologies such as real bot traffic injection, tests in controlled environments, synthetized network traffic, and others. Comparing this framework with others mentioned in the related work section would also be a good challenge and an important step forward.

Even if developed detection systems are not enough to stop botnet activity perhaps there will be a point in time where the investment made on building a botnet won't payoff anymore. This is the greatest motivation of this research. In the end, it will be virtually impossible to detect a botnet whose bots mimic normal host behavior. But a bot behaving as a normal host can't be that malicious.

Many things have now been said and it's time to put some thoughts on the table in order to bring forward some criticism and, hopefully, some discussion and new ideas into this matter.

Besides those attributes with no relevance for this study, some network traffic attributes were not used but should be considered in future researches. This is the case of IPv6 information contained in netflows. This analysis could, and should, be made only on networks with widespread implementation and use of IPv6. For that matter, network equipment should be configured to also collect IPv6 flows.

As said before, the proposed detection framework involved heavy CPU processing and considerable storage needs. The proposed framework could be reengineered to improve its algorithms performance allowing faster and lighter detection cycles. A good streaming algorithm would allow the detection framework to constantly maintain the hosts' diversity values with minimum computation and storage overhead. On the other hand, Random Forests [66] classifiers applied to classification algorithms such as Naïve Bayes and MultiNomial Logit could allow much lighter statistical calculations based on traffic samples and not the entire traffic dataset. Such approaches would need deep validation to be accepted.

Having performance matters in mind, escalation is an issue. One possible approach to escalation would be the deployment of several detection frameworks throughout the network on what could be called *detection-nodes*. This approach would bring forward the need to create *super-nodes*. Super-nodes would be responsible for processing the detections made on *detection-nodes*, horizontally analyzing and crossing all the *detection-nodes* detections.

A system's administrator work is not easy and all tools in hand are simply not enough. To be really useful, a detection framework should allow simple interactions with its operators in order to create tight symbiosis between them both. To contribute to such symbiosis, an integrated front-end could be implemented allowing system administrators to catalog on-line all the threats found in the network. All cataloged threats could then be used by the detections system to support posterior detections.

The proposed model is an anomaly-based approach. It is easy to understand that an ideal detection model would gather all the strengths of the many other known detection approaches. Although falling outside the goal of this thesis, it is easy to see that false positive rates could be reduced if other parallel detections such as those provided by honeypots and signature-based systems were used and crossed.

ROC Plots, related charts, and blacklists used, were useful in the evaluation of the systems sensitivity allowing a better understanding of a system's detection rate evaluation. They are not useful, though, when used alone to evaluate the detection's accuracy or compare it to other detection systems. In order to do so, different, already mentioned, evaluations methods were needed. From real bot traffic usage to Deep Packet Inspection (privacy issues!), different methods can be used to confirm detections and correctly evaluate detection rates.

Not few times IDSs are evaluated according to information theory methodologies without being aware of the true needs of a network administrator. Many good ideas and research paths can be left unattended when pure theoretical approaches are taken for the problems.

Since the beginning of this work one thought kept arising: *Does this make sense?* Or using a Shakespeare's saying: *Is this a wild goose chase?*

The greatest difficulties and normally the deepest pitfalls are the ones that are not visible. In 2009, Ian I. Mitroff and Abraham Silvers, defined statistical errors of type III and IV as "developing good answers to the wrong questions" and "deliberately selecting the wrong questions for intensive and skilled investigation" [68]. Deliberate errors can be avoided supported by one's own ethical values. On the other hand, some apparently obvious decisions can be traps silently waiting for some mind bugs or tricks to fall in. Hopefully all these situations were overcome by retesting good results and posing antagonist questions to the ones already answered.

The final conclusion is that this research vector has many, already mentioned, virtues. Many other similar researches have already been concluded with proven good results.

Gertrude Stein was true: there isn't any there here. The search must go on, but many wider paths are now visible.

8. References

[1] J. Mieres, "Ice IX, The first crimeware based on the leaked zeus sources," *Securelist Website - Kaspersky Labs*, 2011. [Online]. Available: http://www.securelist.com/en/blog/208193087/Ice_IX_the_first_crimeware_based_on_the_leaked_ZeuS_sources. [Accessed: 12-Sep-2011].

[2] C. Mielke and H. Chen, "Botnets, and the cybercriminal underground," in *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on*, 2008, pp. 206–211.

[3] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses," in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, 2009, pp. 299–304.

[4] L. Jing, X. Yang, G. Kaveh, D. Hongmei, and Z. Jingyuan, "Botnet: Classification, attacks, detection, tracing, and preventive measures," *EURASIP journal on wireless communications and networking*, vol. 2009, 2009.

[5] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Mis Quarterly*, pp. 75–105, 2004.

[6] S. Mansfield-Devine, "Anonymous: serious threat or mere annoyance?," *Network Security*, vol. 2011, no. 1, pp. 4–10, 2011.

[7] A. V. Barsamian, "Network characterization for botnet detection using statistical-behavioral methods," Dartmouth College, 2009.

[8] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *Security & Privacy, IEEE*, vol. 1, no. 4, pp. 33–39, 2003.

[9] B. Al-Duwairi and L. Al-Ebbini, "BotDigger: A Fuzzy Inference System for Botnet Detection," in *ICIMP '10 Proceedings of the 2010 Fifth International Conference on Internet Monitoring and Protection*, 2010.

[10] D. Dagon and G. Gu, "A taxonomy of botnet structures," *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, 2007.

[11] J. Oikarinen and D. Reed, "Internet Relay Chat Protocol." 1993.

[12] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.

[13] T. Holz and C. Gorecki, "Measuring and detecting fast-flux service networks," *16th Annual Network & Distributed System Security Symposium*, 2008.

[14] A. Caglayan, M. Toothaker, D. Drapaeau, D. Burke, and G. Eaton, "Behavioral analysis of fast flux service networks," in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research Cyber Security and Information Intelligence Challenges and Strategies - CSIIRW '09*, 2009, p. 1.

[15] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, 2010, pp. 311–320.

[16] J. A. Morales, A. Al-Bataineh, S. Xu, and R. Sandhu, "Analyzing DNS activities of bot processes," in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, 2010, pp. 98–103.

[17] I. Kim, H. Choi, and H. Lee, "BotXrayer: Exposing Botnets by Visualizing DNS Traffic," 2009.

[18] V. C. Estrada and A. Nakao, "A Survey on the Use of Traffic Traces to Battle Internet Threats," *2010 Third International Conference on Knowledge Discovery and Data Mining on Knowledge Discovery and Data Mining*, 2010.

[19] T. Holz, M. Steiner, and F. Dahl, "Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm," in *LEET'08 Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[20] "P2P as botnet command and control: a deeper insight," *3rd International Conference on Malicious and Unwanted Software, 2008. MALWARE 2008.*, 2008.

[21] T. F. Yen and M. K. Reiter, "Are Your Hosts Trading or Plotting? Telling P2P File-Sharing and Bots Apart," in *2010 International Conference on Distributed Computing Systems*, 2010, pp. 241–252.

[22] The Honeynet Project, "Know Your Enemy: Fast-Flux Service Networks," *The Honeynet Project Website*, 2008. [Online]. Available: http://www.honeynet.org/papers/ff. [Accessed: 18-Oct-2011].

[23] B. Stone-Gross et al., "Your botnet is my botnet: analysis of a botnet takeover," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 635–647.

[24] R. Villamarin-Salomón and J. C. Brustoloni, "Identifying botnets using anomaly detection techniques applied to DNS traffic," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008, pp. 476–481.

[25] S. Golovanov and I. Soumenkov, "TDL4 – Top Bot," *Securelist Website - Kaspersky Labs*, 2011. [Online]. Available: http://www.securelist.com/en/analysis/204792180/TDL4_Top_Bot. [Accessed: 12-Sep-2011].

[26] Symantec Intelligence, "Symantec Intelligence Report : September 2011," 2011.

[27] N. Falliere, L. O. Murchu, and E. Chien, “W32 . Stuxnet Dossier,” 2011.

[28] P. Renals and G. Jacoby, “Blocking Skype through Deep Packet Inspection,” in *HICSS ’09 Proceedings of the 42nd Hawaii International Conference on System Sciences*, 2009.

[29] Y. Kugisaki, Y. Kasahara, Y. Hori, and K. Sakurai, “Bot detection based on traffic analysis,” in *ipc*, 2007, pp. 303–306.

[30] Allot Communications, “Digging Deeper Into Deep Packet Inspection (DPI).” 2007.

[31] J. Goebel and T. Holz, “Rishi: Identify bot contaminated hosts by irc nickname evaluation,” in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007, p. 8.

[32] H. Choi, H. Lee, and H. Kim, “Botnet detection by monitoring group activities in DNS traffic,” in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, 2007, pp. 715–720.

[33] A. M. Manasrah, A. Hasan, O. A. Abouabdalla, and S. Ramadass, “Detecting Botnet Activities Based on Abnormal DNS traffic,” *International Journal of Computer Science and Information Security, IJCSIS*, vol. 6, no. 1, p. 8, Nov. 2009.

[34] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han, “Botnet research survey,” in *Computer Software and Applications, 2008. COMPSAC’08. 32nd Annual IEEE International*, 2008, pp. 967–972.

[35] A. Ramachandran, N. Feamster, and D. Dagon, “Revealing botnet membership using DNSBL counter-intelligence,” in *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet-Volume 2*, 2006, pp. 8–8.

[36] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “BotHunter: detecting malware infection through IDS-driven dialog correlation,” in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, 2007, pp. 12:1–12:16.

[37] G. Gu, R. Perdisci, J. Zhang, and W. Lee, “BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection,” in *Proceedings of the 17th conference on Security symposium*, 2008, pp. 139–154.

[38] M. Akiyama, “A proposal of metrics for botnet detection based on its cooperative behavior,” *2007 International Symposium on Applications and the Internet Workshops (SAINTW’07)*, 2007.

[39] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, “Botnet detection based on network behavior,” *Botnet Detection*, pp. 1–24, 2008.

[40] W. Lu and M. Tavallaee, “BotCop: An online botnet traffic classifier,” *2009 Seventh Annual Communications Networks and Services Research Conference*, 2009.

[41] A. Karasaridis, B. Rexroad, and D. Hoeflin, “Wide-scale botnet detection and characterization,” in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007, p. 7.

[42] D. Bolzoni, “Revisiting anomaly-based network intrusion detection systems,” 2009.

[43] Office Of The Privacy Commissioner, “Office of the Privacy Commissioner – Deep Packet Inspection (http://dpi.priv.gc.ca),” *Webpage*, 2010. [Online]. Available: http://dpi.priv.gc.ca/. [Accessed: 15-Nov-2010].

[44] B. Claise, “Cisco Systems NetFlow Services Export Version 9,” 2004.

[45] Sflow Consorcium, “sFlow.org - Making the Network Visible,” 2011. [Online]. Available: http://www.sflow.org/. [Accessed: 03-Nov-2011].

[46] B. H. Trammell, T. Dietz, B. Claise, S. Bryant, and S. Leinen, “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information,” 2008.

[47] Ntop, “Ntop tools webpage,” 2011. [Online]. Available: http://www.ntop.org/. [Accessed: 03-Nov-2011].

[48] Nfdump, “Nfdump tools webpage,” 2010. [Online]. Available: http://nfdump.sourceforge.net/. [Accessed: 03-Jan-2011].

[49] Microsoft, “Microsoft SQL Server 2008,” 2011. [Online]. Available: http://www.microsoft.com/sqlserver/en/us/default.aspx. [Accessed: 05-Jan-2011].

[50] Microsoft, “Microsoft Analysis Services 2008,” 2011. [Online]. Available: http://www.microsoft.com/sqlserver/2008/en/us/analysis-services.aspx. [Accessed: 05-Jan-2011].

[51] Microsoft, “Microsoft Reporting Services 2008,” 2011. [Online]. Available: http://www.microsoft.com/sqlserver/2008/en/us/reporting.aspx. [Accessed: 05-Jan-2011].

[52] Nfsen Project, “Nfsen.” Nfsen Project, 2011.

[53] Perl, “Perl Programming Language.” 2011.

[54] B. Tellenbach, M. Burkhart, D. Sornette, and T. Maillart, “Beyond shannon: Characterizing internet traffic with generalized entropy metrics,” *PASSIVE AND ACTIVE NETWORK MEASUREMENT*, 2009.

[55] E. Simpson, “Measurement of diversity,” *Nature*, 1949.

[56] “Padre, the Perl IDE.” [Online]. Available: http://padre.perlide.org/. [Accessed: 15-Nov-2011].

[57] R. Berthier, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, and D. Sheleheda, "Nfsight: netflow-based network awareness tool," *Proceedings of the 24th USENIX LISA*, 2010.

[58] "Sucuri Security." [Online]. Available: http://sucuri.net/. [Accessed: 15-Jan-2012].

[59] "malc0de.com." [Online]. Available: http://malc0de.com/dashboard/. [Accessed: 16-Jan-2012].

[60] "dshield Home | DShield; Cooperative Network Security Community - Internet Security." [Online]. Available: http://www.dshield.org/. [Accessed: 16-Jan-2012].

[61] "abuse.ch - The Swiss Security Blog." [Online]. Available: http://www.abuse.ch/. [Accessed: 16-Jan-2012].

[62] "Spam IP List Stopping Spams, Scams, and Improving Internet Security Spam-IP.com." [Online]. Available: http://spam-ip.com/. [Accessed: 16-Jan-2012].

[63] VMWare, "VMWare Virtualization Software," 2011. [Online]. Available: http://www.vmware.com/. [Accessed: 15-Nov-2011].

[64] M. P. Collins and M. K. Reiter, "On the limits of payload-oblivious network attack detection," *Recent Advances in Intrusion Detection*, 2008.

[65] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security - ASIACCS '06*, 2006, p. 90.

[66] A. Prinzie and D. Van den Poel, "Random multiclass classification: Generalizing random forests to random mnl and random nb," in *Database and Expert Systems Applications*, 2007, pp. 349–358.

[67] Y. Liu, L. Zhang, and Y. Guan, "Sketch-based Streaming PCA Algorithm for Network-wide Traffic Anomaly Detection," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, 2010, pp. 807–816.

[68] I. I. Mitroff and A. Silvers, *Dirty rotten strategies: how we trick ourselves and others into solving the wrong problems precisely*. Stanford University Press, 2010.

[69] D. Sancho, "You Scratch My Back ... BREDOLAB ' s Sudden Rise in Prominence," *Trend Micro Website*, 2009. [Online]. Available: http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/bredolab_final.pdf. [Accessed: 20-Sep-2011].

[70] É. Young, M. Ballano, and T. Katsuki, "Trojan.Bredolab," *Symantec Website*, 2009. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2009-052907-2436-99&tabid=2. [Accessed: 20-Sep-2011].

[71] G. Tenebro, “The Bredolab Files,” *Symantec Website*, 2009. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_bredolab_files.pdf. [Accessed: 20-Sep-2011].

[72] E. Chien, “Downadup: Attempts at Smart Network Scanning,” *Symantec Website*, 2010. [Online]. Available: http://www.symantec.com/connect/blogs/downadup-attempts-smart-network-scanning. [Accessed: 23-Sep-2011].

[73] Security Intel Analysis Team, “W32.Downadup Infection Statistics,” *Symantec Website*, 2010. [Online]. Available: http://www.symantec.com/connect/blogs/w32downadup-infection-statistics. [Accessed: 21-Sep-2011].

[74] Security Intel Analysis Team, “W32.Downadup.A and W32.Downadup.B Statistics,” *Symantec Website*, 2010. .

[75] P. Porras, H. Saidi, and V. Yegneswaran, “An Analysis of Conficker’s Logic and Rendezvous Points,” 2009.

[76] P. Porras, H. Saidi, and V. Yegneswaran, “An Analysis of Conficker C,” 2009.

[77] P. Coogan, “The Mariposa / Butterfly Bot Kit,” *Symantec Website*, 2009. [Online]. Available: http://www.symantec.com/connect/blogs/mariposa-butterfly-bot-kit. [Accessed: 15-Sep-2011].

[78] Defence Intelligence, “Mariposa Botnet Analysis,” *Defence Intelligence Website*, 2010. [Online]. Available: http://www.defintel.com/docs/Mariposa_Analysis.pdf. [Accessed: 16-Sep-2011].

[79] A. Gostev, “Rustock and all that,” *Securelist Website - Kaspersky Labs*, 2008. [Online]. Available: http://www.securelist.com/en/analysis/204792011/Rustock_and_All_That. [Accessed: 12-Oct-2011].

[80] D. Anselmi et al., “Security Intelligence Report (Win32/Rustock),” 2011.

[81] P. Porras, “A multi-perspective analysis of the storm (peacomm) worm,” *Computer Science Laboratory, SRI*, 2007.

[82] Symantec Security Response, “W32.Duqu: The precursor to the next Stuxnet,” 2011.

[83] S. Mansfield-Devine, “Hacking the hackers,” *Computer Fraud & Security*, vol. 2009, no. 6, pp. 10–13, 2009.

[84] “The Flow of MBR Rootkit Trojan Resumes | Symantec Connect Community.” [Online]. Available: http://www.symantec.com/connect/blogs/flow-mbr-rootkit-trojan-resumes. [Accessed: 12-Oct-2011].

[85] H. Binsalleeh et al., “On the analysis of the Zeus botnet crimeware toolkit,” in *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, 2010.

[86] N. Falliere and E. Chien, “Zeus : King of the Bots,” 2009.

9. Apendix A: Botnet History

Analyzed botnets are presented bellow in the form of subsections. In the analysis below, a study was made on known bot behaviors and actions not intending to be a full disclosure of botnet's activities. Functionalities and behavior analysis presented for each botnet are relative to information available at the time of this study not being constant through time. Bots do whatever they're instructed to and some extra downloads can enhance dramatically bot functionality.

9.1.1 BredoLab (Oficla)

This botnet study was based on the analysis made by Trend Micro Incorporated in [69] and Symantec in [70] and [71].

The BredoLab malware family uses drive-by-download[9] and email as its preferred proliferation channels into Windows operating systems. Drive-by-download infection of a host is achieved by the herder using social engineering applied to malicious webpages and browser or browser plugins vulnerabilities. Plugins such as ActiveX, DirectShow, Flash and PDF are some of the plugins normally affected by vulnerabilities in the Windows operating systems.

Email proliferation of the bot is achieved by widespread email spam sent to numerous users. Every email can include a link to a malicious website in order to achieve a drive-by-download infection or include a malicious payload of its own.

Initially built as a dropper for the Zeus bot (described ahead in this section), Bredolab malware evolved to achieve the botnet status on its own though being used nowadays to install other malware and bots. Among installed malware the following threats can be found: Backdoor.Haxdoor, Backdoor.Rustock, Downloader.MisleadApp, Hacktool.Rootkit, Infostealer, Trojan.FakeAV, Trojan.KillAV, Trojan.Pandex, Trojan.Srizbi, W32.Koobface, W32.Waledac.

[9] In this thesis context "drive-by-download" means download of software from the internet without known consent from the user. This can happen when the download happens without the user cognizance or when the user allows the download and installation but is not aware of the implications of such action. The first definition involves browser vulnerabilities (for example); the last definition involves user deception by a malicious webpage or email (for example).

Bredolab botnet is known to be built on centralized HTTP C&C topologies with some of its servers installed in hacked legitimate websites. Fast-fluxing domains used to obfuscate the botnet operations were also found in the researches from Trend Micro and Symantec. In the fast-fluxing operation, several IP addresses were observed to be assigned to one single domain thus allowing larger bandwidth and load-balanced communications between bots and C&C servers. To create a bigger mesh, each IP address was assigned, in turn, to many malicious domains.

In order to achieve longer malware lifecycles, the Bredolab malware family attempts to inject itself into *explorer.exe* and *svchost.exe* processes of Windows operating systems in order to avoid detection by firewalls and antivirus.

Bredolab bots are also known to use encoded communications and include Virtual Machine detection capabilities to difficult botnet operation analysis. Researchers often use virtual machines in order to debug and analyze bot behaviors. To difficult researcher's analysis, some bots are capable of detecting that they are running on virtual machines automatically shutting themselves down.

Since vast panoply of malware can be installed on an infected host, the attack vectors are endless. Studied Bredolab attacks can involve installation of additional backdoors as well as fake antivirus scareware. Fake antivirus scams are used to trick the owners of the infected PC's into buying a complete, but fake, antivirus suite.

9.1.2 Conficker (DownUp, DownAndUp, DownAdUp, Kido)

All data presented here related to the Conficker network and worm was based on the reports [72], [73] and [74] by Symantec Corporation and [75] and [76] by SRI International.

Conficker worm targets the Windows operation systems using the exploitation of Windows Server Services vulnerabilities through buffer overflows in Remote Procedure Calls (RPC) as one of its infection vectors.

Specific shell code executed through RPC buffer overflow contacts a malicious webserver in order to download and attach the full worm DLL to a system process in the target host. The worm is also able to deploy its own webserver allowing the target machine to contact itself in order to download the referred binary or DLL. Other means of Conficker proliferation include the use of removable media and visible Netbios shares to spread the virus DLL. The use of visible but protected network shares is achieved by the worm through the use of dictionary attacks.

C&C communications make use of Domain Fluxing and Domain Generation Algorithms (DGA) in order to establish and hide its infrastructure. DGA generates 250 pseudo-random domain addresses each day. Control servers also allow download of bot updates and additional malware. In an ad-hoc P2P topology, the UDP protocol is used to maintain a list of the infected and TCP to transfer data between the bots. Conficker can also use named-pipes to exchange URLs for downloadable payloads between infected hosts.

High availability and long malware lifecycles are achieved by the use of encrypted payloads and communications, changing system patches to allow and control reinfection, deleting Windows restore points, disabling some important Windows services such as Automatic updates, Security Center or Windows Defender, auto installing as a Windows service and blocking determined DNS lookups.

Similarly to Bredolab, Conficker installs W32.Waledac Spambot allowing botmasters to send large amounts of unsolicited emails through infected hosts therefore promoting more infections and host recruitment.

Another known attack is related to the SpyProtect scareware. This scareware tricks the user into installing a fake antivirus program and asking to pay an amount for an upgrade.

The use of DGA can also produce some side-attack-effects. Though this was not thought in the bot design, the Conficker DGA algorithm is expected to produce some real domains. The use of a real domain by the botnet can pushup web requests on those domains possibly causing a DDoS on the servers hosting such domains.

Like in all known P2P and DGA based botnets takedowns, reverse engineering the domain generation Algorithm was the keystone to proactively blacklist the malicious domains being created every day. Additionally, the registration of a possible update domain allowed researchers to detect three million unique IP addresses trying to get an update over the course of one week. [73]

9.1.3 Mariposa

All the data gathered and analyzed regarding the Mariposa Botnet was based on the findings reported in [77] and [78] by Symantec and Defence Intelligence. Defence Intelligence was one of the companies involved in this botnet dismantlement.

In order to recruit new bots, the Mariposa botnet controls and monitors Windows Messenger (MSN) and Explorer processes. Monitoring MSN process allows Mariposa to inject malicious links into messages exchanged between users. On the other hand, controlling the Explorer process makes it possible to control removable media (USB) infections.

Another known Mariposa infection vector is the dissemination of malicious files in P2P networks. Botmasters or bots themselves publish infected files in a P2P network and then make other bots to subscribe those files. This black-hat SEO strategy enables such malicious files to rank high in torrent search engines raising download probabilities and possible infections ratio.

Looking at the bot's own manual, Mariposa is able to inject code and monitor the Windows Explorer process being also capable of encoded communications. [77]

C&C is based on several control servers and domains using encrypted UDP datagrams for its information exchange. Mariposa bots are also able, under instructions, to download other forms of malware such as Blackenergy DDoS bots. Mariposa is also capable of self-updating from available file sharing websites such as Rapid Share.

Mariposa's known attacks capabilities involve cyber scamming, DDoS – TCP (SYN) and UDP flood, Data theft (passwords, bank credentials and credit cards) and email SPAM sending.

9.1.4 Nugache

All the Nugache botnet information presented in this section was based on the findings described in [20].

Back in 2006, Nugache was easily identified by antivirus software because of its week deception capabilities. Nugache communications, for instance, used invariantly TCP port 8 turning its bots into sitting ducks for antimalware software hunts. Initially built on IRC-based C&C, Nugache was then upgraded to start using P2P encrypted communication channels.

Scan behaviors were also observed in Nugache operations since bots tried to propagate through exploitation of remote vulnerabilities in Microsoft services such as the Local Security Authority Subsystem Service (LSASS) and Distributed Component Object Model (DCOM) service.

Other propagation vectors also used by the Nugache bot were SPAM emailing using the Windows Address Book (WAB) as well as sending forged messages with malicious links to user's contacts in instant messengers such as AIM and MSN.

A more peculiar form of propagation involving social engineering and black-hat SEO has also been observed in Nugache distribution. For that matter, hackers and herders started their job by creating a variant of a popular free application website and hosting a malicious variant of that application. The second step was to register the modified application and website in freeware download aggregators. The herder then instructed the bots in the already established Nugache network to request the download of the malicious file just to make it rank high in the applications with most downloads. With higher ranks in freeware aggregators higher probabilities in getting host infections were achieved.

Among Nugache attack capabilities were found two distinct DDoS systems. One based in HTTP GET flooding and other using UDP flooding.

9.1.5 Rustock - RKRustok, Costrat

All the information presented and related to the Rustock botnet is based on two reports. One from Kaspersky Labs [79] and another from a Microsoft response team composed by experts of the Microsoft Digital Crimes Unit, the Microsoft Trustworthy Computing and the Microsoft Malware Protection Center ([80]).

Rustock is known to be a very complete suite of malware components. Its versions of rootkit-enabled backdoor trojans were carefully developed in order to distribute large quantities of spam email. Rustock is best known as a SPAM Botnet.

Along with Rustock, Harnig (a known Rustock dropper) download and installs additional malware in the target host in order to increase host infection time and availability.

Increased availability and longer malware lifecycles are also achieved by the use of bot code compression and obfuscation as well as communications encryption. This bot's rootkit component monitors some Windows events in order to filter itself out of any requests to the system that includes its own name. Beyond windows events hooking, disk and network operations are also concealed by monitoring network and disk system drivers.

Rustock is also capable of checking the presence of kernel debuggers (WinDBG, Syser and SoftICE) in order to difficult analysis of its code. In addition, it checks itself for modifications using CRC32 checksums.

SPAM email dissemination capabilities changed throughout Rustock's lifetime. Initial versions of Rustock used a built-in SMTP client engine in order to send spam. Rustock was then changed, in 2008, to send spam through Windows Live Hotmail. This fact allowed Microsoft response teams to get the crucial law warrant that made possible this botnet takedown. The switch to Hotmail was made because the process of sending email from a user's personal computer was clear evidence and sign of a malware infection. This behavior was easily detected or blocked by firewalls and other monitoring tools. The Hotmail switch brought another advantage to Rustock: the use of SSL. Initial email messages were sent in clear text. Using Hotmail allowed Rustock to encrypt its traffic using HTTPS.

Beyond SPAM attacks, Rustock was found redirecting traffic to malicious websites selling rogue security products using a scareware scam tactics. Rustock was also able of installing such rogue security products.

Rustock C&C was designed with a built-in fallback mechanism. Normal C&C behavior was to connect to known, fixed, C&C servers. In case such servers were unavailable, Rustock design included a DGA mechanism to generate, on a daily basis, 16 new domain names that bots could contact to get their commands. Some variants of the DGA algorithm have been found in different Rustock versions sometimes allowing the generation of 96 new domain names each day.

Rustock takedown was achieved on March 16, 2011, when servers were seized from hosting providers in the United States following a Microsoft injunction. This injunction was the result of combined Microsoft, U.S. federal law enforcement agents, FireEye, and the University of Washington efforts to stop the Rustock botnet. The takedown was only possible with the help of Dutch authorities and the Chinese Community Emergency Response Teams (CERT) that helped to dismantle part of the botnet structure implemented outside the borders of the United States of America.

From the collected servers, it was able to conclude that some cyber-attacks into Russian IP space were started from those servers. It was also possible to see that Rustock C&C servers were supported by money sent through an online payment service account. The account used had an address in Moscow associated with it.

Rustock alone was capable of sending 30 billion spam email messages per day. In its good days it was responsible for about one third of the world SPAM sent. In January and February 2011, it was detected that more than 1,300,000 unique IP addresses belonging to the Rustock botnet were connected to the internet performing SPAM attacks.

9.1.6 Storm - Dorf, Nuwar, Peacomm, Peed, Tibs, Zhelatin

The Storm botnet analysis was based on the research done in [19] and [81].

Though propagating solely through spam, Storm was recognized by its smart and evolved social engineering tricks. All malicious emails used English written texts with topical, actual and varied content such as recent news, disasters or holidays. Along with well written texts, links to malicious websites were included inviting the email addressees to click them. Malware droppers downloaded to vulnerable hosts were found to be polymorphic meaning that the downloaded malware programs changed their signature on every download.

After starting its work, the malware dropper, installed the bot code as well as an encrypted configuration file containing information related to other botnet peers available. Each peer was defined by a hash value and an IP/Port. This information was enough for the bot to join the P2P botnet.

In order to maintain inter-bot synchronization, Storm bots used Network Time Protocol (NTP) to forcibly update infected hosts time

Storm's C&C was already mentioned in previous sections. For C&C communications and control servers obfuscation, Storm used an encrypted version of the OVERNET P2P network, a P2P distributed hash table routing protocol based on the Kademlia protocol.

Increased botnet resilience was also achieved by constant bot updates in order to adapt to OS upgrades, improved heuristics in antivirus products, and security patches. Besides upgrades, Storm malware also used rootkit approaches.

9.1.7 Stuxnet

The study presented in this section regarding the Stuxnet botnet was based on the complete Symantec report in [27].

Stuxnet and its variant Duku [82] are completely different botnets from the ones analyzed so far in this thesis. Differences come out when observing attack targets as well as infection and proliferation methods.

The Stuxnet botnet stands out from other botnets regarding attack targets. Stuxnet was designed and programmed having in mind control over specific industrial control systems. For that matter it included functionalities that allowed it to modify code on Programmable Logic Controllers (PLCs). PLCs are widely used in industrial facilities such as factories and power plants in order to implement and control industrial processes.

By the time the report in [27] was elaborated, it was Symantec's believe that Stuxnet's target was likely located in Iran since almost all detected infections were located there. Stuxnet bots auto-destruct themselves when the infected host's IP was not geo-located near Iran. Looking at the infection process, Stuxnet's goal was certainly to sabotage an Iranian industrial facility.

Although Stuxnet operations were discovered in July 2010, deeper analysis revealed that it must have existed since 2009. This fact shows that the operation behind this botnet was timely planned. It also comprehended a deadline to reach its target. The final date found in Stuxnet bot configuration files was June 24, 2012. That meant that, after such date, the malware would cease to operate.

Stuxnet included in its weapon's arsenal several zero-day exploits supporting various infection processes based on network, removable media and others types of dissemination routes. Reliability was improved by evolved rootkits (both for Windows OS and PLC) that included antivirus evasion and process injection techniques.

Beyond exploitation of many network Windows Services vulnerabilities in order to spread its infection, Stuxnet searched for Siemens SIMATIC Step 7 industrial control software and a specific vulnerability in the access to its WinCC database server. The WinCC vulnerability exploited using SQL injection was not the only attack on Step 7 software. Stuxnet was also able to infect Step 7 projects that ran malicious code when the project was loaded.

A specific aspect of removable media infection shows that the bots were made to keep the botnet's footprint as reduced as possible: USB infection would delete itself from the USB key after the third infection. This behavior differs from other kinds of infection that try to reach as much targets as possible.

In order to masquerade its infections, Stuxnet's designers were able to sign their code using compromised certificates such as the ones from Realtek Semiconductor and JMicron Technology Corps (both later revoked).

Whenever Stuxnet bot was executed, it compared its own version with the version available in the C&C server and upgraded itself if necessary. This specific upgrade process was complemented by a P2P update mechanism that ran within local area networks (LANs).

Encrypted C&C communications included data sent by infected hosts to control servers such as infected host's IP addresses, OS versions, and information regarding the presence of Siemens SIMATIC Step 7 industrial control software. C&C topology was based on simple HTTP control servers.

9.1.8 TDL (TDSS, Alureon)

This botnet study was based on the analysis published in [25] by Kaspersky Labs experts on version 4 of the botnet.

As other covered malware, TDL achieved host infection through driven-by-downloads and malicious email links and attachments.

TDL is a C&C peer-to-peer based botnet using the Kad Network[10]. Herders created an encrypted file in the Kad Network containing a list of commands for the bots. Bots then downloaded and executed issued commands.

Bot resilience was improved using known methods such as encrypted network connections. Encryption protected infected computers from network traffic analysis as well as blocked other cybercriminals' attempts to take control of the botnet. It also blocked access to Windows Update services, disabled antivirus products and used a rootkit component in order to hide its presence from host user.

TDL was designed with a real antivirus embedded being able to remove several malicious programs such as Gbot, Zeus, Clishmic or Optima. Beyond malware removal capabilities, TDL blacklisted addresses of other botnet's C&C servers thus disrupting their communications.

[10] Kad network is a peer-to-peer (P2P) network which implements the Kademlia P2P overlay protocol.

TLD was also found to install other malicious programs such as fake antivirus, adware, and SPAM dissemination support software (Pushdo spambot, for example).

TLD attack vectors also included redirection of search engines in order to commit click fraud. Moreover, TLD bot was able to intercept network traffic capturing usernames, passwords and credit card data.

9.1.9 Torpig - Sinowal, Anserin

The resources used to analyze the Torpig botnet and its related malware suite can be found in [23], [83] and [84].

Torpig is related to the Mebroot rootkit [84] and both were designed to target the Windows operating systems family. The purpose of its creators was to steal bank and credit card data.

The Torpig trojan injected its code into windows processes such as *services.exe* turning off antivirus applications. Since it could also provide remote access to the infected machine through the use of backdoors, it allowed the botnet herders to later install other kinds of malware.

A group of researchers at the University of Santa Barbara in California decompiled the Torpig trojan in order to dissect its C&C communications. Researchers were able to install a controlled C&C server interacting with part of the Torpig botnet [23].

Since Torpig uses a domain fluxing approach (domains of the C&C servers are constantly changing) researchers had to reverse engineer the trojan’s Domain Generation Algorithm (DGA). Torpig’s DGA revealed to be based on current date and Twitter Trends API.

With complete DGA in hand, researchers were able to generate and register a domain ahead of botnet herders taking partial control of the botnet. The partial control of the Torpig botnet lasted for 10 days. In that period, researchers' server was contacted by 182 800 bots (estimated value). Among all the data received from bots, there were credentials of 8 310 PayPal and bank accounts, as well as 1 660 credit and debit card numbers [23].

9.1.10 Zeus (Zbot, PRG, Wsnpoem, Gorhax, Kneber)

The Zeus botnet analysis was based on the paper published in [85] as well as in the Symantec report in [86]. The Zeus bot or Zbot is a malware package that has been available for sale or rent in the cybercrime undergrounds since 2007.

The peculiar aspect with Zbot was that it was delivered as a software builder allowing malware authors to create their own custom bots and control servers very simply. The Zbot programming package contained a builder that could quickly generate a bot executable as well as Web server files (PHP, images, SQL templates) to use as an HTTP C&C server.

The primary purpose of Zbot, as other bots, was financial gain through data theft of online credentials (FTP, email, banking, and other). Online credentials were stolen from infected hosts by compromising protected storage information or monitoring network communications looking for FTP and POP3 passwords sent in clear text.

Zeus bots were also capable of gathering operating system information and execute additional actions specified in its configuration file or in control server's task lists. Host infections were achieved using the popular methods already described before: drive-by-downloads and email SPAM.

If included in its configuration file, Zbot performed some advanced attacks such as web page injection. Online banking services implement authentication processes that try to avoid key logging and network sniffing. In order to avoid such countermeasures, certain malware evolved to use web page injection. In web page injection, a malicious process monitors the user's web page accesses. When interesting pages (online banking or payment services for example) are accessed by the user, the malicious process injects additional malicious HTML or javascript code to the page accessed. Injected code can, for example, require user credential not actually required by the real site.

Zbot implemented commands such as reboot and shut down infected hosts, delete system files, initiate backdoor connections, execute remote code, block and unblock URLs access as well as upload and download files.

Fingerprinting was also done by this bot's malware family. Upon start, it was capable of sending a multicast UDP query in order to search the network for UPnP devices such as routers. Zbot also had the ability of reconfiguring broadband routers in order to extend its malicious capabilities.

Zeus botnet used encrypted HTTP C&C communications. All data was sent using the RC4 encryption algorithm and a key specified by the botnet author.

In order to hide its malicious presence, Zeus bot was able to detect antivirus and firewalls canceling bot installation if determined anti-malware products were running. It was also capable of changing the Windows registry to restart the installation process between operating system restarts as well as injecting itself into the Winlogon process. Resilience was even further improved by copying itself to a hidden file in the Windows\System32 directory.

Since Zbot code became available openly in the World Wide Web, many variants of this malware appeared with improved features such as domain-fluxing and use of DGA techniques. Ice IX [1], Murofet and LICAT are some of them.