Identifying, Modeling and Detecting Botnet Behaviors in the Network

Thesis · November 2014

DOI: 10.13140/2.1.3488.8006

1 author:

Sebastián García
Czech Technical University in Prague
35 PUBLICATIONS **46** CITATIONS

Universidad Nacional del Centro de la Provincia de Buenos Aires

Doctoral Thesis

Identifying, Modeling and Detecting Botnet Behaviors in the Network

Author:
Sebastián García

Advisor:
Dr. Alejandro Zunino

CoAdvisor:
Dr. Marcelo Campo

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Computer Science

in the

Instituto Superior de Ingeniería de Software Tandil
Departamento de Computación y Sistemas

Friday 28th November, 2014

UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

Abstract

Facultad de Ciencias Exactas
Departamento de Computación y Sistemas

Doctor of Philosophy

Identifying, Modeling and Detecting Botnet Behaviors in the Network

by Sebastián García

Botnets are the technological backbone supporting myriad of attacks, including identity stealing, organizational spying, DoS, SPAM, government-sponsored attacks and spying of political dissidents among others. The research community works hard creating detection algorithms of botnet network traffic. These algorithms have been partially successful, but are difficult to reproduce and verify; being often commercialized. However, the advances in machine learning algorithms and the access to better botnet datasets start showing promising results. The shift of the detection techniques to behavioral-based models has proved to be a better approach to the analysis of botnet patterns. However, the current knowledge of the botnet actions and patterns does not seem to be deep enough to create adequate traffic models that could be used to detect botnets in real networks. This thesis proposes three new botnet detection methods and a new model of botnet behavior that are based in a deep understanding of the botnet behaviors in the network. First the SimDetect method, that analyzes the structural similarities of clustered botnet traffic. Second the BClus method, that clusters traffic according to its connection patterns and uses decision rules to detect unknown botnet in the network. Third, the CCDetector method, that uses a novel state-based behavioral model of known Command and Control channels to train a Markov Chain and to detect similar traffic in unknown real networks. The BClus and CCDetector methods were compared with third-party detection methods, showing their use in real environments. The core of the CCDetector method is our state-based behavioral model of botnet actions. This model is capable of representing the changes in the behaviors over time. To support the research we use a huge dataset of botnet traffic that was captured in our Malware Capture Facility Project. The dataset is varied, large, public, real and has Background, Normal and Botnet labels. The tools, dataset and algorithms were released as free software. Our algorithms give a new high-level interface to identify, visualize and block botnet behaviors in the networks.

Acknowledgements

Esta tesis es la conclusión y también el comienzo de una parte muy importante de mi vida. Y como todo en la vida, nunca hacemos las cosas solos. Tuve la suerte de hacer esta tesis sobre los hombros de mucha gente que me dio su trabajo, cuidado, amor, enseñanzas, guía, ayuda y soporte para que la pudiera terminar. Les agradezco eternamente a mis papas, que dedican buena parte de sus vidas a asegurarse de que sea feliz. Ellos son la fuente de lo que soy. El hijo del amor, la energía y el cuidado; la continuación de una vida de enseñanzas y apoyo. Ellos están en todo lo que hago. Gracias por darme una hermosa vida. Esta tesis está dedicada a Vero, mi amor y mi pareja. Gracias por cada momento compartido, cada risa y cada llanto, cada proyecto y cada idea loca que hacemos juntos. Tu apoyo me dio seguridad, tu comprensión me permitió seguir, tu soporte me estabilizó, tu estímulo me dio fuerzas y tu amor me llenó y enseñó. Caminando de la mano esta vida con vos, soy feliz. Como parte de mi vida, mi hermana forma parte de mi personalidad y sin ella no estaría acá. Gracias por todas las enseñanzas, los cuidados, todas las veces que te tomé de la mano, todo el soporte y el amor. Tu guía es parte de mi. Y gracias a mis sobrinos Santi y Tomy, que llenan mi corazón.

Desde el primer día de mi doctorado, mi director Alejandro Zunino estuvo ahí para mí. Su guía fue invalorable, comprendiéndome y soportando los errores. Gracias por aceptarme y creer en mí. Comprendí que cada doctorando necesita algo diferente, gracias por darme libertad. Gracias a mi co-director Marcelo Campo por abrirme las puertas y darme la oportunidad para hacer el doctorado. Gracias especialmente a CONICET por posibilitar que me dedique a la investigación. Gracias a todo el equipo del ISISTAN, que me guió, corrigió y apoyó. Especialmente a mis amigos en el ISISTAN, que soportaron tantas visitas y me dieron un apoyo cuando lo necesitaba. Agradezco también a Roberto Giordano Lerena, que desde su rol en UFASTA me apoyó, confió y dio la libertad para hacer el doctorado. Gracias también a mis alumnos en UFASTA que fueron parte de los comienzos de mi trabajo y tanto nos divertimos. Una parte vital de mi trabajo fue ir a la Universidad CTU en Praga a terminar mi doctorado. Esto fue posible gracias al esfuerzo y apoyo de Michal Pechoucek, Martin Rehak y Martin Grill. Gracias por confiar en mí. Fue también un placer compartir un hermoso proyecto con Martin Grill, Honza Stiborek y Harpo. Gracias a mi alumno en la CTU Vojtěch Uhlíř, que hizo un muy buen trabajo en los dataset. Gracias a mis amigos de la CTU que me hicieron sentir como en casa trabajando ahí. También quiero agradecer a Osvaldo y Mari, por sus palabras y guía; y porque me enseñaron a respirar. Finalmente agradezco a mis amigos, que estuvieron ahi, apoyando, aconsejando y escuchando, muchas veces

a la distancia. No sería el mismo sin ustedes. Gracias Lucia, Dario, Sole, Ingrid, Tadeo, Eve, Petr, Karel, Lea Tami, Lea Ferrari, Maxi, Luciano, Gaby, Vir y Vico.

This thesis is the conclusion and also the beginning of an important part in my life. And as everything in life, we never make things alone. I was lucky enough to make this thesis upon the work, care, love, teachings, guidance, help and support of many people along the way. I specially thank my parents, who devote a good part of their lives to make sure I'm happy. They are the source of who I am. The son of love, energy and caring; the continuation of a life of teachings and support. They are a part of what I do. Thanks for giving me a wonderful life. This thesis is dedicated to Vero, my love and partner. Thanks for each shared moment, each laughter and tears, each project and crazy idea we plan together. Your support gave me confidence, your understanding allowed me to continue, your backing gave me stability, your encouragement gave me strength and your love filled my heart and taught my soul. It makes me happy to walk this life hand in hand with you. As part of my life, my sister helped forge my personality. I wouldn't be here without her. Thanks for all the teachings, the caring, each time I hold your hand, all the guidance, all the support and love. And thanks to my beautiful nephews Santi and Tomy, that fill my heart.

From the very first day my advisor Alejandro Zunino was there for me. His guidance was invaluable, understanding me and supporting my errors. Thanks for accepting me and believing in me. I understood that each student needs something different, thanks for giving me freedom. Thanks to my co-advisor Marcelo Campo for opening me the doors and giving me the chance to start the PhD. Special thanks to CONICET, for allowing me to research. Thanks to all the ISISTAN team, that guided, corrected and supported me. Specially to my ISISTAN friends that endured so many visits and gave me support when I needed it. I also want to thank Roberto Giordano Lerena, because his support and trust gave me the freedom to start the PhD. Thanks also to my students in UFASTA, that were part of this from the beginning. A very important decision in my PhD was to move to Prague and work in the CTU University. This was possible thanks to the effort and work of Michal Pechoucek, Martin Rehak and Martin Grill. Thanks for your trust. It was also a pleasure to share a wonderful project with Martin Grill, Honza Stiborek and Harpo. Thanks to my student in CTU Vojtěch Uhlíř for his amazing work in the datasets. Thanks to my friends in CTU University that make me feel at home working there. I would also like to thank Osvaldo and Mari, for their words of support and because they taugth me to breath. Finally I would like to thank my friends. You were there for me, encouraging, guiding and listening, many times distantly. I would not be the same without you. Thanks Lucia, Dario, Sole, Ingrid, Tadeo, Eve, Petr, Karel, Lea Tami, Lea Ferrari, Maxi, Luciano, Gaby, Vir and Vico.

Contents

List of Figures

List of Tables

1

To Vero. My partner, my friend, my love.

Chapter 1

Introduction

In the last decade botnets have evolved from being used as a personal activity platform to becoming a financially aimed structure controlled by malicious groups [1][2]. A botnet is a network of remotely controlled, compromised computers, used for malicious purposes. Hosts in a botnet are called *Bots* and the owners of a botnet are called *Botmasters*. From small DDoS (Distributed Denial of Service attacks) to world wide spam campaigns, botnets have become the technological backbone of a growing community of malicious activities [3] and remain as the most significant tool that threatens the Internet today.

Botnets are engaged in a large number of activities for the attackers community. Their use is far larger than the visible SPAM being sent and the known DDoS. As a huge overlay network, botnets are a complex support platform for attacking others by maintaining control over a large number of computers. The complete set of actions is unknown, but they are known to be used to steal private data and financial information, to attack others with DDoS, to commit click fraud, to send SPAM, to be part in APT (Advanced Persistent Threat) attacks, to monitor users and political entities, to abuse the resources in the infected computers, etc. These large and different number of actions suggest a vast and complex infrastructure which behavior is not easy to underpin.

The security research community has tackled the problem of detecting botnets from many perspectives, but most of them can be categorized inside two types: host-based detections and network-based detections. Among the host-based techniques are the traditional antivirus, antimalware and antispyware software that tried to detect the malicious binary files inside a computer. Among the network-based techniques are the IDS (Intrusion detection systems) and IPS (Intrusion prevention systems) that analyze the packets in the network. Both types of systems are meant for different purposes

and usually complement each other. The most important advantage of the host-based techniques is that they can stop the infection before it occurs, and the most important advantage of the network-based methods is that they can protect a huge number of computers at the same time. This thesis is among the network-based analysis and detection techniques.

Traditionally, the detection methods applied statical techniques to detect botnets [4]. However, with the increasing complexity of botnets actions and attacks they shifted to behavioral-based models [5][6] in an attempt to deal with the changes over time [7]. These newer behavioral-based algorithms seem to have different degrees of success. Unfortunately it is difficult to know exactly how good they are because most of them are impossible to reproduce due to the lack of a complete description [8], correct methodology [9] or the lack of a good botnet dataset [10]. These deficiencies prevent researchers from using them in any comparison. Moreover, most of these methods were commercialized and are therefore not available for the research community.

Looking at the current state of the botnet detection area we ask ourselves how can we develop a botnet detection method that uses behavioral models of network traffic and that is able to correctly identify them in real networks.

1.1 Motivation

The evolution from the first viruses, trojans and worms to the current botnets has been fast and complex. The main difference of botnets is not on the way they spread or attack but on the communication and control topology they implement. The differences between a virus, a worm and a trojan are on how they infect a binary file or on how they spread themselves. However, the first distinctive characteristic of a botnet is that each infected host is part of an overlay network. The botnets brought a new way of controlling and managing a large number of infected computers. It may be hypothesized that the growth of the Internet, the bandwidth and the amount of computers gave the attackers a new problem to solve: how to control, coordinate and update a huge amount of infected computers continuously?

It is often underestimated that to be considered malware, a key characteristic is to be malicious. If there are not malicious intentions, then it is not malware. However, it could still be a botnet, because being a botnet implies a new way of communicating and not a way of infecting. In fact there are several benign botnets on the Internet, although not called botnets. The most famous is perhaps the Boinc project [11]. This

is a volunteer computing [1] project where the user can choose to which internal work, e.g. mathematics, physics, alien life, etc., she will donate her computing resources. Unfortunately, the term botnet is now closely related with malicious activities.

The first theoretic approach to build and control a network of infected computers was supposedly included in the first worm ever released on the Internet on 1988: the Morris worm [12]. It was reported that the original source code had comments about how to send commands and to control the infected computers remotely. Fortunately, the features were not implemented. Ten years had to pass until the first public report of a malicious botnet on the wild in 1999 [13] talked about a worm using an IRC network for communication. From this moment, the already known malware implements communication protocols that allow their owners to control all the infected computers at the same time. These mechanisms are called command and control channels (C&C).

The first C&C channel implemented on malicious botnets was done using the IRC protocol. This protocol was selected because the IRC community had implemented for a long time benign bots to control the chat channels. One of the earliest benign bots was Eggdrop, which was designed to administer several IRC channels with several Eggdrop implementations and to allow all the Eggdrop implementations to communicate. Possibly without intention, the first benign botnet was born [14]. Using the power of the IRC protocol several botnets were created and some of them are still working in 2014. The communication topology of the IRC-based C&C channels is considered as centralized. The botmaster, or owner of the botnet, makes all the bots connect to the same IRC server and channel and therefore she is able to control the bots at the same time or one by one. The power to simultaneously control all the bots has special relevance to fulfill Distributed Denial of Service (DDoS) attacks. The IRC topology had a special characteristic (that was lately lost with other C&C topologies such as HTTP) that the server was able to pushes commands to the bots. In IRC, the server is continually asking if the bot is alive, and the bot is answering if it is alive. When the server wants to send some command, it just push it to the IRC channel and all the bots receive it. We can say that the command was pushed from the C&C server to the bots. In later C&C topologies this behavior was changed to the bots asking commands to the server (the commands are being pulled).

The IRC bots were part of an important switch in the malware community toward more malicious activities. The first versions of the Agobot [15] botnet in 2002 implemented malicious but very limited actions. However, researchers reported [14] on the new Spybot botnet, two years later, that "(...)New exploits were integrated into the code only days after proof-of-concept was published, they began to openly attack anti-virus

[1] https://boinc.berkeley.edu/trac/wiki/VolunteerComputing

and firewall software. They ran proxies for spam relaying, allowed remote control of Webcams, allowed hackers to sniff the network the infected machine sat on, ran open SOCKS proxies, logged keys, stole passwords and target IM account information, overwrote hosts files to prevent access to key corporate AV/Firewall websites, ran HTTP and FTP servers, served files using DCC, opened remote command shells, controlled DNS and ARP records, even come with their own rootkits.".

With time, the botmasters became aware of the limitations of the IRC protocol. For example, if someone took down one IRC server the botmaster looses all the bots. Also, one bot could sniff all the communications of the rest of the botnet and therefore allow researchers to learn about the operations. The botmasters changed the protocol used in favor of P2P (Peer to Peer) protocols [16][17]. However, the first usage of P2P (Peer to Peer) networks by botnets was only to spread the binary file and not as part of the communication channel [18]. Some versions of the Spybot botnet placed the binary file in the shared folder of other P2P programs in order to accomplish the distribution.

The P2P protocol was possibly used by malware for the first time as part of the Apache Scalper worm on 2003 [19]. In that case the P2P protocol was inefficient and rudimentary, but still working. Since then, different implementations of the P2P protocol were used to transfer the infected binaries, to maintain the C&C channel, and to communicate between peers. The P2P protocols have clear advantages over the previous approaches: they can be decentralized, they can hide the amount of bots infected, they can avoid being sniffed and they can be highly resilient to take down attempts. The final strength of the P2P botnets rest in the type of P2P protocol implemented and the design of the bot code.

Since 2005, researchers had witnessed another shift in the C&C topology toward the HTTP protocol. This protocol gave botnets several advantages over the previous ones. The two most important characteristics are that the botnet traffic can be successfully hidden inside normal HTTP traffic and therefore avoid detection, and that the Internet is almost entirely made of normal Web servers that the botnet can infect and use as part of its infrastructure. The HTTP protocol considerably expanded the capabilities of botnets into highly resilient structures. Nowadays most of the botnets use the HTTP protocol for their C&C and malicious activities.

Apart from the C&C topology used, botnets use several security measures to assure they can reach the C&C servers. The two most common techniques are the use of Domain Generation Algorithms (DGA) [20] and Fast-Flux techniques [21]. DGAs are a way of generating new Internet domain names. The bot has a presumably secret algorithm that generates a new domain name from time to time. The bot tries to contact each domain and to establish a C&C channel. If the domain is down, the bot continues

with the next generated domain. When the Botmaster wants to control the botnet all she has to do is to register the next domain that she knows will be asked next. In the case of Fast-Flux networks the technique basically consists in changing the IP address registered for a domain name very quickly. In this way the real server that is part of the botnet is difficult to find because there are thousands of intermediary servers changing continuously.

There has been also some examples of botnets focusing on particular social networks such as Facebook [2] and Twitter [3] [22], however since the websites belong to a unique company it has been easier to shut down the botnets.

It is very difficult to estimate statistics about the botnet phenomenon. While some reports may give us an idea of how big they are, other may report how active they are or how many attacks they are performing. The Shadowserver foundation [4] reports that in the last two years there have been up to 2,700 different and active C&C servers at one single moment. The number of C&C servers keeps varying and the servers keep changing. However, in the last three years the amount of active bots has been between 50,000 and 150,000, which is a large amount for only one monitoring institution. An anonymous Botmaster has declared in a Reddit IAMA post that only his botnet has more than 12,000 bots [5].

The constant growth of the botnets and malware in general is fueled by the economical possibilities of such activities [23]. The underground economy behind the creation and administration of botnets has many participants and organizations, from starting teenagers to government supported teams. The goals of these organizations are also diverse, ranging from personal vengeance to the steal of trade secrets and the harassment of political parties. Some of the possibilities of earning money with a botnet that were found in the underground [23] are: Rent parts of the botnet for money (The tenant may use it for their own purposes), send SPAM for money (Some company pays for having its advertisement being sent), selling of private digital identities, selling of credit card information, click fraud, pay-per-download, DDoS for hire, extortion, bitcoin mining, etc.

Apart from these activities, the underground economy has other means of earning money that are not strictly related to botnets, such as the pay-per-install (PPI) [24] commoditization. The PPI service is the most used option for installing your malware for the first time in thousands of victims.

[2] http://www.facebook.com
[3] http://www.twitter.com
[4] https://www.shadowserver.org/wiki/pmwiki.php/Stats/BotnetCharts
[5] http://stratumsecurity.com/wp-content/uploads/2012/06/yxMDx.jpg

Botnets take control of the victims using several different mechanisms. The most common infection vector in the past years has been the drive-by-download [25], a method where the victim has to download (willing or not) a piece of malware that is then executed in her computer. Usually this means malware coming in an email or most probably malware being downloaded by accessing an infected web site. Once this first malware is installed, the botnets start a chain of updates and installations to leave the victim computer ready for usage. The goals of the updates are to install certain malware modules that are needed for operation, like new C&C channels, SPAM modules, DDoS modules, among others. These modules are usually bought from third-party malware developers, since it is difficult for one botmaster to develop all the parts of the botnet [24] itself. Once the botnet is working and active, the control of the victim is done using a C&C channel.

Botnets change continuously [26]. They are deep and complex structures designed to avoid detection, to survive take down attempts, to update their controls mechanisms, to change their attacks and to even remain idle for some time in order to go undetected. The botmasters are part of a large group of organizations that systematically and methodically change their tactics based on what the security research community is doing in the botnet detection field. The detection of botnets is a slow but steady changing game where each new defense method is overcome by a new attack method. It is a cycle of defense-avoidance actions that is meant to change over time. This complexity if the main motivation of our study.

1.2 Our Proposed Botnet Detection Approaches

The previous research done in the area of botnet detection suggests that by analyzing the dynamic behavior of the botnets in the network it may be possible to obtain good detection results [27]. It is based on the idea that the behavior-based techniques can deal better with the complex actions of botnets over time. The results achieved depend on the type of network where the analysis is being made, on the type of botnets being analyzed, on how the errors are accounted and most importantly on how the method represents the botnet behavior.

As botnets keep changing, evolving and evading detection methods, it is not possible to come up with a unique and static solution which error metrics remain significant over time. The best botnet detection methods that were deployed ten years ago, cannot detect the current botnets. Therefore, new dynamic and adaptable methods are needed.

This thesis hypothesize that it is possible to build behavioral models of the botnet actions on the network that can be used to detect the infected hosts with improved error metrics over the current methods. These models can be built upon a deep and careful analysis of the botnet actions that identify each distinct botnet behavior.

The main goal of our thesis is to **implement a behavioral-based botnet detection method by analyzing the network traffic**.

The plan of our thesis includes several sub-goals that are necessary to achieve our main goal:

- To understand the behaviors of botnets by analyzing real traffic.
- To compare the results with other methods; including a new comparison methodology and a new error metric.
- To capture a large, varied, real and labeled dataset of botnet, background and normal traffic.
- To make our software, algorithms and datasets freely available to the community.

To progress in the goal of understanding the behavior of botnets it is first necessary to capture and analyze real botnet data. Having a good dataset is the first step. A first option may be to use a third-party and public dataset, however, there are currently no public datasets that fulfill all the conditions that we need, that is: that have real traffic from botnets, at least three types of labels and it is a long capture. Another not so common option is to simulate the botnet behavior based on some models of what botnets may do [28], however, these simulations cannot mimic the real behaviors of botnets and therefore can usually not be used to learn or train a generic algorithm. Consequently, our best option was to get our own botnet captures. Obtaining real botnet data is a difficult task because it needs a special laboratory setup, a special Internet connection, it is time-consuming and most importantly needs real and active malware binaries. Apart from the botnet data, a good dataset has to have also normal data. However, to obtain normal data is, if possible, more difficult than obtaining botnet data. The problem is that no one is willing to public the private information that is included in the traffic of a normal user. Moreover, the normal behavior of a user cannot be automatized to generate more data, because it is not normal any more. The last part of a good dataset is the Background traffic. This traffic is important because it is unknown and it could be captured in any network researchers have access to. Therefore, one important task that we keep improving during all our work was to create a real dataset of botnet, background and normal traffic. Our dataset is now huge and is still

continually growing, but it started as a collection of home-made malware captures. This dataset may be one of the most important parts of our thesis, because it gave us the clues to create our detection methods. It is completely described in Chapter 8, but it is used by all of our methods along the thesis.

To fulfill our research goals we designed three different botnet detection methods. Each method addresses a specific part of the botnet problem and contributed to our understanding of the botnet actions. Each method is more complex than the previous one and therefore advances a little more in the resolution of the problem.

Our three detection methods do not use the packets in the network but instead they use flows. A flow is a conceptual structure that aggregates and summarizes all the packets sharing the following five fields: source IP address, source port, destination IP address, destination port and protocol. By working with flows we have all the information about the packets (except they payload) in a more condensed and meaningful structure. In our first detection method, we separate the flows in one second time windows, and then we aggregate for each time window the following features: amount of different source IP addresses (*sips*), amount of different destinations IP contacted (*dips*), amount of different source ports used (*sports*) and amount of different destination ports contacted (*dports*). Then, we end up with a single register having four numbers per each time window. The basic idea behind this design is the concept that the bots connections may have a similar group of values, such as for example having larger *dips*. The aggregated data is then clustered and each final cluster is analyzed to see how many botnet and normal traffic were included. The main limitation of this approach is that there is no automatic detection of botnets. This approach was tested with our first dataset and successfully proved that it is possible to detect an infected bot by aggregating the traffic in a network.

Our first detection method is called SimDetect 3. It was used as a learning step on how botnets behave. The behavioral feature that we tried to detect was the fact that the connection pattern of one bot alone looked very similar to the connection pattern of each of the other bots. This means that it may be possible to distinguish between an infected computer and a non-infected computer because all the infected ones behave similarly in the network. The SimDetect method uses time windows to separate the traffic and then clusters certain features.

Our experience with the SimDetect method [29] made us realize that we needed to improve the dataset with better botnets. We also realized that we needed to design a better detection method. In order to improve the dataset we started a collaboration with the CTU University in Prague, Czech Republic, to design, develop and publish a large dataset of botnet captures. At the same time we started our second detection proposal, called BClus, based on the lessons learned during the capture of the dataset.

The dataset created together with the CTU University consists in thirteen different captures. Each capture represents a specific botnet scenario and it includes a specific botnet family, the background traffic of a complete university department and the traffic of near fifteen normal computers. The totality of the thirteen captures took up almost 700 GB of space. This complete, varied and real dataset was manually labeled and published in Internet. To the best of our knowledge it is the most complete botnet dataset published so far [30].

Our second detection method is called BClus 4. The main idea was first to separate the traffic in bigger time windows, then to aggregate the traffic of each host separately instead of all the IP addresses together, then to compute some features of the aggregated data and then to clusterize the aggregated information searching for similar groups of behavior. Each cluster, unlike the previous approach, contains the behavior of each IP address. To automatically detect the clusters that represent a botnet behavior, we extracted a new group of features for each cluster and then we trained a rule-based classification algorithm to learn how to find those clusters. All the training and testing was possible due to the labeled dataset created with the CTU University.

The results obtained with the BClus method [30] were encouraging and better than in the SimDetect method. However, to have a real measurement of how good the results were, we compared our algorithm with other two third-party detection methods [30]. The first one was the main anomaly-based detection method of the CTU University, called CAMNEP [31]. This method is a large ensemble of more than 20 different algorithms that is implemented as a ready-to-sell detection solution. CAMNEP implements some of the most advanced anomaly detection algorithms published and is actively used in several organizations. Fortunately, the research team of CAMNEP was part of the collaboration and therefore we could run this method on our dataset. The second third-party algorithm used in the comparison was BotHunter, a well-known detection algorithm published by Gufrei Gu. et al. [32] that is one of the most popular botnet detection methods in the community.

A key part of the comparison with other methods was to coordinate and design how the methods were going to be measured against each other. The question we asked ourselves was: Which is the best way of knowing which method is the best? The traditional way of doing this is to compute the errors based on the amount of flows that were correctly detected, and later use some ratios of these errors, such as True Positive Ratio (TPR) or F-Measure (FM). Most of the time, this classic error metrics work well, but in our research they are not enough to take into consideration the needs of a network administrator. Therefore, we created a new error metric that may be better to compare botnet detection methods [30]. Its first difference is that it works with IP address instead

of flows. This allow us to know if an IP address was correctly detected or not, regardless of its flows. The second difference is that it works with time windows for computing the error metrics. By taking the time into consideration it is possible now to assign a weighted value to the errors and differentiate between detecting a botnet earlier or later, which is a good way to know if a method is better. The combination of IP addresses and time results in a new set of error values called weighted-TruePositives, weighted-TrueNegatives, weighted-FalseNegatives, weighted-FalsePositives and a new set of error metrics such as weighted-TPR or weighted-F-Measure. This is the error metric that was used in all our comparisons.

The results of the comparison of the BClus method with the third-party methods showed that for the thirteen captures in the dataset, our method had significantly better error metrics than the rest. However, every comparison should be carefully analyzed. The most important consideration to be done is that the CAMNEP method was configured to be used in real environments were the customers do not want to have a large amount of False Positives, so CAMNEP had few True Positives and therefore also a smaller F-Measure. As far as we know this was the first comparison of botnet detection methods on a real, labeled and large botnet dataset with normal and background labels.

After the successful comparison of the BClus method, we were aware of the limitations of our technique and we end up having a good knowledge of how the botnets worked in the network. This knowledge made us realize that, first, the botnet traffic had certain *states* that could be used to modeled their behavior. Second, that these models could be later used for detection, and third, that we needed to get richer botnet datasets. Therefore, we started design our third detection method, called CCDetector, based on these needs.

During the creation of the BClus method we also started a new project called the Malware Capture Facility Project (MCFP). The MCFP is a medium scale setup of virtual machines that allow us to continuously infect more than 30 computers for long periods of time. We were able, then, to capture and analyze the behavior of botnets during months. The dataset generated by this project is public and is a fundamental part of our knowledge about botnets.

The CCDetector method, described in Chapter 6, has two parts. First, it models the traffic using the changes in the states of the connections. Second, it represents these models with a Markov Chain that is used for detecting and finding similar traffic in a new network. The first part uses three basic features of each flow to represent the behavior: the size of a flow, the duration of a flow and the periodicity of a flow. We realized that each state of a connection could be represented by a combination of these three features. Therefore, we discretized the values of these features in three ranges

using thresholds and obtained a total of 36 different states. Every time a new flow is received, one of these states is assigned to it. In this first part, instead of only working with the flows in the network, we start working with a novel aggregation of flows that we called *4-tuples*. A 4-tuple is composed of all the flows that share the same source IP address, the same destination IP address, the same destination port and the same protocol. Then, a 4-tuple identifies all the connections coming from the botnet to a specific *service*. The motivation to create the 4-tuples was a deep understanding of the behavior of the C&C channels. It is commonly believed that the flows of a C&C channel are periodic. However, most of the times when a botnet generates a new connection to a C&C server (for example every 30 minutes) the operating system is assigning a new source port for the packets. This means that despite that the botnet *is* actually generating new connections to the same port every 30 minutes, we would obtain a *new* flow per connection and therefore we cannot compute the periodicity of one flow. In consequence, our 4-tuples representation forgets the source port and are able to aggregate all the corresponding flows together. With all the flows being aggregated by 4-tuple and each flow receiving a new state, we have now for each 4-tuple a chain of states that represents its evolution over time.

In the second part of this method, we create a Markov Chain that models the transitions in the states of chains of each 4-tuple. It is worth noticing that each Markov Chain represents a 4-tuple and therefore represents a specific behavior of the botnet and not all the botnet traffic at once. These Markov Chains are the core part of the model and can be stored for future usage in detection phases.

During the testing of the CCDetector method, the models that were created during the training are used to detect similar botnet behaviors in a new network. First, we capture the new flows in each 4-tuple and we assign a states to each flow. This new chain of states is then compared to each of the Markov Chains already stored for the trained models to see which one has the greatest probability of generating that particular chain of states. Once the winner model is found, the label of the stored model is assigned to the flows in the new 4-tuple.

Based on our previous experience comparing the BClus method, we also compared the CCDetector method to three third-party detection methods: the CAMNEP method, the BotHunter algorithm and the BClus method. The results show that we were able to greatly improve the error metrics of our previous comparison and in consequence we were able to better detect the botnet traffic. With this third method we conclude the presentation of our novel techniques.

In summary the main contributions of this thesis are:

- A proposal to detect the bot traffic of one single host. (Chapter 3)
- A proposal to detect botnets by clustering network patterns. (Chapter 4)
- A new error metric to compare botnet detection methods. (Chapter 7)
- A new model for representing the botnet behaviors in the network. (Chapter 6)
- A proposal to detect the botnet traffic based on the changing states of the C&C traffic. (Chapter 6)
- A new, large, labeled and public dataset from the Malware Capture Facility Project. (Chapter 8)

In conclusion, we developed three different botnet detection methods based on a large amount of labeled botnet traffic and we compare our results with three other proposals. In all the comparisons our method showed better error metrics, proving that it was possible to improve the current results in the area by deeply studying the botnet traffic.

1.3 Thesis outline

The thesis is organized to show the most important achievements in a logical and almost chronological order. This chapter made an introduction to the botnet detection topic and described the main goals and results in this thesis. The rest of the work is organized as follows.

Chapter 2 presents a survey on the most important and current botnet detection methods. It has first an analysis of previous surveys in botnet detection, then a deep description of each proposal and a final discussion of the open problems in the botnet detection area. The chapter also proposes a new topology map of botnet detection methods.

Chapter 3 describes the SimDetect method for detecting a single bot by looking at the similarities in the traffic. Some botnet captures were done to support it.

Chapter 4 shows the BClus method for detecting botnets by analyzing the traffic of all the hosts in the network and clusterizing the similarities between the botnet traffic. It uses a semi-supervised algorithm.

Chapter 5 shows our deep analysis of one example botnet during 57 days of operation. This analysis is the basis of understanding the complexity of a botnet and to learn how to detect the states of a botnet.

Chapter 6 shows the CCDetector method for detecting botnets by using the changes in the state of a botnet. The changes were modeled as a Markov Chain and can be stored on disk for later use. This method takes advantage of our deep understanding of the behavior of the botnets in the network.

Chapter 7 describes the comparison effort between the BClus method, the CCDetector method and two more third-party detection algorithms. A new error metric is presented to compare botnet detection methods. The dataset used is real, large, varied and labeled.

Chapter 8 deeply describes the datasets used in our methods and explains the design and creation of our Malware Capture Facility Project, that it is used to create botnet datasets continually.

Chapter 9 presents the conclusions and lessons learned during this thesis.

Appendix A deeply describes the behavioral characteristics of the dataset used in the CCDetector method.

Appendix B briefly shows the details of the programs that we implemented for our detection methods. Most of them were published online.

Chapter 2

Previous Work

Researchers have been working in the area of botnet and malware detection since the first automatic SPAM message, that was sent on April 12th, 1994 in the USENET network [33]. The efforts done in the botnet and malware detection area have generated a vast amount of proposals that analyze the problem from multiple perspectives. There have been also some surveys that study and classify the botnets papers. The goal of this chapter is to study how the previous proposals detected botnets and which were their motivations, datasets and results. We later study which the shortcomings of these proposals are and how this thesis intends to fill those gaps.

This Chapter analyzes, classifies and compares the most relevant network-based botnet detection methods to date. It is divided into four major parts. First, an analysis of the previous surveys on botnet detection methods. It is a view on how other surveys had organized the detection proposals. Second, a new classification and comparison of the detection proposals. Each aspect of the botnet detection proposals is compared and summarized. Third, a deep analysis of the most important detection proposals. Each proposal is fully described to understand its operation context. Fourth, an analysis of the most important issues found in the area; where the shortcomings are analyzed.

2.1 Previous Surveys on Botnets

Apart from the botnet detection proposals themselves, there have been also previous surveys and summaries of these proposals. Before getting into the details of the detection methods it is useful to study how others have attempted to order and classify the proposals. As far as we know, there are no previous surveys on botnet detection

methods. But there are surveys on *botnets*. However, since several surveys on botnets include a brief analysis of detection methods, this Section analyzes them.

2.1.1 Description of Previous Surveys on Botnets

A simple classification of detection methods was done in [34] using two categories: honeynets and passive traffic. Regarding the detection methods, the focus of the survey is on how to track the botnets. From this point of view, the survey identifies the honeynets as a widely used method for tracking botnets by studying their actions inside the system. The passive traffic class is identified as a generic category for all the network-based methods that capture packets in the network. Beside these categories, the paper does not present any other analysis of detection methods.

Several data sources for botnet detection are enumerated in [35]. Also, a separation between detection techniques and measurem studies is proposed. The analysis of behavior is included in the last group. In addition, the survey uses a table to relate data sources, proposed techniques found in the literature and invariant bot behaviors. However, these last behaviors are not deeply analyzed, and the final organization is rather confusing. For example, the detection techniques of the comparison table have no relation with the detection techniques previously discussed in the same paper. Finally, the invariant botnet properties proposed (propagation, communication and attack) are not described or defined.

The detection techniques are classified into four classes in [36]: signature-based, anomaly-based, Domain Name System (DNS)-based and mining-based techniques. This is the first study to use capabilities in a comparison table of detection techniques: ability to detect unknown bots, capability of botnet detection regardless of botnet protocol, encrypted command and control (C&C) channels and structure, real-time detection and accuracy. Some of these are included in this chapter for comparison. The main contribution is the description of four detection classes and the analysis of more than thirteen papers.

The evadability of detection methods is studied in [37]. The evasion cost is proposed as a measure of how good each method is. This cost represents the complexity of the evasion technique and the utility lost by the botnet when the evasion technique is successful. Eight detection characteristics are proposed: Basis, Hub, Internet Relay Chat (IRC), Flow-chars, Time, Net-Det, Syntax and Taint. These characteristics are the most complete detection methods topology presented to date.

A discussion on how to use traffic measure methods to deal with network security issues is presented in [38]. This is the first study that describes the current limitations of the intrusion detection field. It warns against four common problems: simulated datasets that are not good enough, lack of normal traffic models, detection features that tend to be overfitted and a few validity and reliability problems in the evaluation criteria. It is not included in the survey comparison table of this chapter because it does not compare detection methods.

Several botnet detection and tracing methods are analyzed in [39]. They are separated into honeypot-based, IRC-based and DNS-based methods. The IRC-based category is separated into traffic analysis-based and anomaly activities-based methods.

A comprehensive botnet topology is presented in [40]. It includes infection mechanisms, C&C models and detection methods among others. However, the analysis only distinguishes between signature-based algorithms and anomaly-based algorithms.

Botnet attacks and threats are analyzed in [41]. It states the importance of discovering abnormal behaviors. These behaviors are categorized as network based, host based and global correlated. Unfortunately, no further analysis of these behaviors is performed.

A topology of network-based and anomaly-based detection systems is presented in [42]. Detection types are classified into learnt models (where normal behavior representation is obtained automatically) and specification models (where normal behavior representation is obtained manually). Learnt models detect anomalies by comparing new traffic against three cases: rules of normality, models of normality and normality statistics profiles. The models of normality use data mining, neural networks or time series analysis, among others. The specification models create models of normal network behaviors on the basis of how protocols are normally used (protocol based), different protocols states (state based) and which protocol transactions are legal (transaction based). This survey is not particularly applied to botnets, but it is closely related to common techniques in the botnet detection area.

The detection methods are separated into honeynet-based and passive traffic monitoring in [43]. Passive traffic monitoring includes the behavior-based, DNS-based and data mining detection. Furthermore, it expands the behavior-based detection to include signature-based and anomaly-based detection.

The detection sources are separated into honeynets and intrusion detection system (IDS) in [44]. IDS sources are separated into signature-based and anomaly-based categories. The anomaly-based category is separated into host-based and network-based

categories. The network-based category is separated into active monitoring and passive monitoring. This is the only survey that proposes a classification for active botnet monitoring.

After this analysis we are now able to compare the characteristics of the surveys.

2.1.2 Comparison of Previous Surveys on Botnets

For the purpose of comparing the previous surveys, it was first necessary to find out how each of them categorized the botnet detection methods. However, it was not possible to find a common categorization criteria because each survey emphasized on different aspects of the papers. Therefore, we have to extract the comparison structure of each survey to understand and summarize the main *dimensions* used to categorize the detection proposals. We used these *dimensions* to compare the previous surveys and show the differences between them. The following are the dimensions extracted from the previous surveys:

- *Detection sources*: It means how the survey deals with the main source of the information used for the detection. How the survey classifies the sources. For example, it can be application logs, NetFlow logs from a private network or network packets from a honeypot.
- *Detection features*: It is related with how the survey classifies the features used for detection. The features are usually used to make a type of topology. For example, encrypted botnet detection, botnet protocol detected and syntax needed for detection.
- *Detection techniques*: It is related to how the survey classify the techniques used for detection. It is more related with the assumptions made about which are the best ways of detecting the botnets. For example, using the behaviors, static fingerprints, anomalies or signature.
- *Detection algorithms*: It is related to how the surveys classify the algorithms used to obtain results (e.g., Bayesian statistics and neuronal networks).

These dimensions are the broader and most general comparison that can be done of the surveys. They were the basis of the comparison of surveys that is shown in Table 2.1. In this Table it can be seen that there is no unified criteria for describing and classifying the detection proposals.

Survey	Detection Sources	Detection Features	Detection Techniques	Detection Algorithms
[35]	Network packets, DNS logs, darknet and traffic flows	Not included	Behavior (attack and cooperative) and signature	Not included
[37]	Not included	Net-det, Syntax, Taint, Time, Basis, Hub, IRC and Flow-chars	Not included	Not included
[36]	Not included	Unknown bots, protocol, encrypted, real time, accuracy and Net-det	Signature, anomaly, DNS and mining based	Not included
[39]	Honeynets and network packets	Not included	Signature and anomaly based	Not included
[40]	Not included	Not included	Signature and anomaly based	Not included
[34]	Honeynets and network packets	Not included	Not included	Not included
[41]	Not included	Not included	Behavior (network and host based and global correlated)	Not included
[42]	Not included	Not included	Anomaly models (model, rule and statistical based) and specification models (protocol, state and transaction based)	Data mining, neuronal networks, pattern matching, expert systems, Bayesian, covariance, matrices, chi-squared
[43]	Honeynets and network traffic	Not included	Behavior, DNS and data mining based	Not included
[44]	Honeynets and IDS	Not included	Signature and anomaly (host and network based)	Not included

TABLE 2.1: Comparison of previous surveys on botnet detection methods

The analysis of the Table 2.1 revealed some limitations on the classification criteria of the surveys. These limitations were a good guide to design our own comparison of detection proposals presented in Section 2.3. The following are the limitations found in the previous surveys:

- All the surveys use a different terminology, which makes the comparison difficult. What a survey calls method is called technique in another. What a survey calls bot is called botnet in another. Moreover there are no definitions of the terms.
- Most surveys focus on few dimensions or do not have dimensions at all. Whereas one survey includes the botnet sources dimension, another only describes the botnet protocols. The labels included labels in Table 2.1 show this issue.
- Most surveys tend to describe the papers rather than analyze them. What a botnet detection paper stated as truth is not even doubted in other surveys.
- Most surveys do not reference enough papers to support their comparison structure.

These limitations suggest the need for a new, broader and up-to-date study. A comprehensive topology and comparison are needed to deeply understand each detection proposal. The next section describes the details of our proposed topology.

2.2 Classification of Previous Botnet Detection Proposals

The previous comparison of surveys help us design a good comparison of botnet detection proposals. Since the goal of this chapter is to better understand how the proposals detected the botnets, we propose three different points of view to make the comparison. The first point of view is a classification of the different *algorithms*, *techniques* and *sources* of the proposals. The idea is to broadly separate all the different options shown in the proposals analyzed in this Chapter. This point of view is represented in the topology described in Subsection 2.2.1. This topology map was already published [8], and as far as we know, it was the first one presented in the network-based botnet detection area. The second point of view used a deeper classification that is related with the evaluation of the complete paper. This is not only related with the method per se, but with all the decisions included in the paper. The second point of view is represented on several tables on Subsections 2.2.2 and 2.2. The third point of view is a deep description of every step in the detection method and is meant to understand the context on which the proposal was made. This third point of view is done in Subsection 2.4.

2.2.1 Topology Map of Network-Based Botnets Detection Characteristics

The first comparison perspective is a type of topology map where each paper can appear more than once. The map, shown in Figure 2.1, aims at presenting a clear view of how the detection proposals differ from each other. It allows researchers to quickly find which papers implemented each technique.

Note that in Figure 2.1, we used [45] to reference the Appendix B of [45]. Also, note that some categories do not have any paper assigned. They were included because they represent well-known techniques and help to see where new papers could be assigned.

The topology map is divided into the following categories to cover the different aspects of the papers:

- Detection algorithms

  It differentiates between the type of algorithms.

  - Supervised: infers a function from supervised (labeled) training data, also known as classifiers.
  - Semi-supervised: uses both labeled and unlabeled data for training.
  - Unsupervised: finds hidden structures in unlabeled data.
  - Signal processing: analyzes signals (filtering, correlation, spectrum analysis, pattern recognition, etc.)
  - Heuristics rules: usually based on ad-hoc techniques, such as manually finding the best threshold.

- Detection techniques

  It differentiates the main technique used for detection.

  - Anomaly based: uses anomaly-based techniques to detect behavior patterns.
    * Bot behavior: detects the behavior of one bot with data from one bot alone.
    * Botnet behavior: detects the behavior of a group of bots acting as a botnet (not individually).
    * Temporal behavior: detects behavior changes over time; involves time measurement.
      · Peer to peer (P2P)
      · Hypertext Transfer Protocol (HTTP)
      · IRC

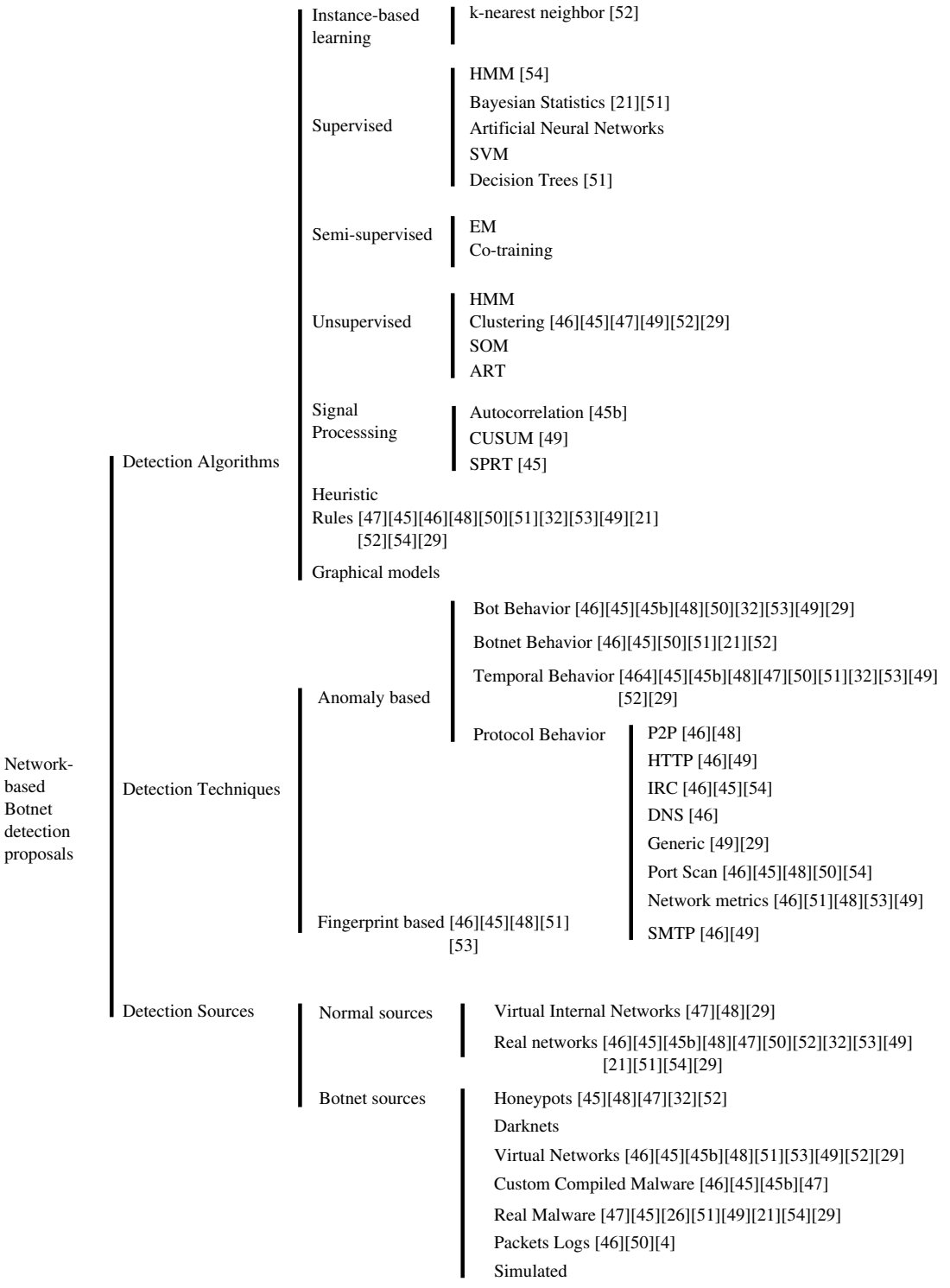

FIGURE 2.1: Topology map of network-based botnet detection characteristics. ART, adaptive resonance theory; CUSUM, cumulative sum; DNS, Domain Name System; EM, expectation–maximization; HTTP, Hypertext Transfer Protocol; HMM, hidden Markov model; Internet Relay Chat; P2P, peer to peer; SOM, self-organizing map; SMTP, Simple Mail Transfer Protocol; SPRT, sequential probability ratio test; SVM, support vector machine.

            - DNS
            - Generic: detects any protocol.
            - Port scan
            - Network metrics: uses network metrics (e.g., bytes per second and packets per minute)
            - Simple Mail Transfer Protocol (SMTP)
        - Fingerprint based: uses a string or byte sequence for detection.

- Detection sources

  It refers to where the packets were captured and not how they were captured.

    - Normal packets: verified normal packets.
        - Virtual internal networks: captures verified normal packets from an internal and controlled network.
        - Real networks: captures packets from real networks; traffic should be labeled background if it is not verified.
    - Botnet packets: refers to the source of botnet packets.
        - Honeypots: honeypots connected to the Internet.
        - Darknets: from darknets.
        - Virtual networks: from a controlled virtual network. Malware is manually installed.
        - Compiled malware: modified versions of known malware.
        - Real malware: uses unmodified real binary malware to infect their computers.
        - Packets logs: uses packet logs.
        - Simulated: simulated data.

The topology map presented is useful in visualizing how each proposal tackled the botnet detection problem. However, it is not enough to understand what each paper proposed. The next subsection uses another point of view to describe the classification of papers in the topology.

2.2.2 Desired Properties of a Botnet Detection Proposal

The botnet detection proposals have a common set of properties that can be used to better understand them. For example, they can be compared on the basis of how they verify the dataset, how they filter the data or how they report results. Most of the

detection proposals have focused on the description of the detection method and on the report of true positive metrics. However, most of them did not consider the importance of other properties, such as the verification of the training dataset, the analysis of the error metrics, the description of the assumptions or the diversity of the training dataset. These are equally or more important than reporting positive results.

This subsection describes what we consider as the 20 most important desired properties of a network-based botnet detection proposal. These properties help to understand and compare each proposal. They show what is missing on each paper and what is being overemphasized.

These properties were created partially based on ideas from previous surveys. They are separated into five groups for better visualization and description. The following are the property description:

- First group: verification issues Verification methods applied to the datasets.
  - Training dataset diversity: how diverse are the botnet training data.
  - Training dataset verification: how is the botnet and normal training data verification performed.
  - Validation dataset verification: how is the botnet and normal validation data verification carried out.
  - Reproducibility: can the proposal be verified and reproduced?
- Second group: results issues Results achieved in the paper, how they were obtained and expressed.
  - Experimental setup: how were the experiments designed?
  - Accuracy-based performance metrics: how well did the proposal perform?
  - Results comparison: were the results compared?
  - Capture in a host or in a network: does it need the proposal to capture from a network or a single host?
- Third group: theoretical background Hypothesis and assumptions in the paper.
  - Paper hypothesis: how was the hypothesis verified?
  - Assumptions: which assumptions were made?
- Fourth group: detection characteristics The main characteristics of the detection.
  - Statical bot detection: which bot statistical features does this proposal detect?

  - Unknown botnet detection: does it detect unknown botnets?
  - Encrypted botnet detection: does it detect botnet that use encrypted connections?
  - Real-time detection: does it detect botnets in real time?
  - Protocol-dependent features: does detection depend on a protocol?

- Fifth group: detection method The detection method itself.

  - Preprocessing: how was the dataset prepared?
  - Main detection method: which is the main detection method proposed?
  - Differentiation from other attacks: how were other attacks, such as port scanning, differentiated?
  - Malicious actions detected: which malicious actions does it detect?
  - Automatic botnet identification method: how are botnets automatically detected?

The next subsection uses these properties to analyze and compare papers.

2.3 A Comparison of Previous Botnet Detection Proposals

This subsection compares the detection papers on the basis of the two previous perspectives. Tables are used for the main topics to help visualize the differences.

The comparisons performed in the following subsections are a good resource to quickly understand the motivations and contexts of the papers. They give hints about the details of the works and help to better understand the proposals.

Before the comparison, a common confusion about the use of the terms bot detection and botnet detection should be addressed. The difference becomes important when dealing with detection methods. It can be safely assumed that usually the methods needed to detect a whole botnet are different from the methods needed to detect one infected computer alone. When we try to detect a bot alone, there is less traffic to analyze, and there is no synchronization or correlation with other bots. Although the detection techniques can be similar, the design and goal of the techniques are different.

2.3.1 Comparison of Anomaly-based Behavior Detection Techniques

Table 2.2 shows the classification of papers in the anomaly-based subcategory of the detection techniques category of the topology map. Data in this table were found after an in-depth analysis of the proposals, because most of them did not show this information explicitly. The relevance of each value in Table 2.2 is different. For example, in N-gram [46], the features are computed within a time interval, and thus, it is considered that it uses temporal behaviors. However, time windows are used only to avoid the continuous computation of the features, and thus, they are not so important. In contrast, in BotHunter [32], time windows are a major part of the proposal. Without them, it would not work properly.

Table 2.3 clearly shows which protocol is used on each proposal.

2.3.2 Comparison of Detection Sources

Tables 2.4 and 2.5 show a comparison of normal and botnet data sources for detection. Each proposal trains or verifies its methods using one or more datasets. The design of the capture methodology and the origin of these datasets are very important to understand the conditions under which the method was validated.

In these tables, the term compiled is used with reference to binary bots compiled by the authors from public source codes. Compiled malware has to be configured before being used. For example, the C&C server and the encryption passwords must be set. Consequently, these malware are not equal to the ones in the wild. However, the implications of using modified malware are normally underestimated. Most proposals do not describe the modifications carried out. The use of custom-compiled malware can be a good approach if the limitations are clearly stated.

These tables also use the term virtual with reference to the use of virtual machines or virtual networks to execute the binaries. The use of virtualization technology is commonly accepted for malware execution, but it should be noted that some malware detects virtual machines to avoid being executed. This could potentially bias the type of malware that the proposal could analyze.

Finally, the terms training and testing are used with reference to the training and testing phases of the method, respectively. When only one of these labels appear, it means that the paper did not consider or need the other phase.

The analysis of Tables 2.4 and 2.5 shows that most proposals lack some type of dataset. If an algorithm needs be trained, at least independent training and testing datasets

Paper	Bot Behavior	Botnet Behavior	Temporal Behavior	Protocol Behavior
Botminer [47]	Packets/flow, mean bytes/packet; connects to SMTP; ask MX records	Orders to every bot; synchro-nization; similar bot traffic pattern	Mean bytes/second and flows/hour	Port scan; ask MX records; packet and byte transfer; fph, ppf, bpp and bps
BotSniffer [45]	Binary download; SPAM	Synchronization; IRC synchronized	Attacks in the same time window; IRC responses in the same time window	Port scan detection
BotHunter [32]	Attack signature prior knowledge	None	Time windows	Port scan detection
Appendix B of [45]	Connects to C&C	None	Connects to HTTP C&C periodically	None
N-Gram [46]	None	None	Features/time interval	Features/protocol
Stability [48]	P2P flows are stable in average bytes/flow	None	Botnet flows stable/time window	Botnet P2P protocol stable
Models [49]	Attack signature prior knowledge; traffic patterns	None	Features analyzed in time slices	Eight traffic features
Unclean [50]	SPAM and phishing	Infect the same unclean networks	Bots appear in the same networks	Port scan detection
FluXOR [21]	None	Anomalies in the features of domains	None	None
Tight [51]	None	Bots controlled simultaneously	C&C sends orders simultaneously	bpp, bps and pps
Tamed [52]	None	Infect the same OS; same C&C; same payload	Time windows	None
Incre mental [53]	Same traffic in different time windows	None	Similar traffic in different time windows	None
Markov [54]	None	None	None	Port scan detection
Synchro nize [29]	Bots synchronize their traffic	None	Traffic is aggregated in time windows	None

TABLE 2.2: Comparison of anomaly-based behavior detection techniques.

Papers	HTTP	IRC	P2P	Generic
BotMiner [47]	✓	✓	✓	-
BotSniffer [45]	✓	✓	-	-
BotHunter [32]	-	✓	-	-
Appendix B [45]	✓	-	-	-
N-Gram [46]	-	✓	-	-
Stability [48]	-	-	✓	-
Models [49]	✓	✓	✓	-
Unclean [50]	-	-	-	✓
FluXOR [21]	-	-	-	✓
Tamed [52]	✓	✓	-	-
Tight [51]	-	✓	-	-
Incremental [53]	-	✓	✓	-
Markov [54]	-	-		✓
Synchronize [29]	-	-	-	✓

TABLE 2.3: Comparison of the Botnet protocols detected.

Papers	Normal Packets
BotMiner [47]	Testing: 10-day university campus
BotSniffer [45]	Testing: university campus IRC; university campus complete
BotHunter [32]	Testing: university campus, 100 Mb/s (two captures); production public/17 network (10 days)
Appendix B [45]	Testing: one HTTP-only capture (17 GB); four full data capture (33 GB)
N-Gram [46]	Training: 1-h Wi-Fi traffic Testing: 1-h Wi-Fi traffic; one virtual IRC (60 clients); one real IRC public network
Stability [48]	Training: four complete university; two virtual P2P traffic Testing: Same as training
Models [49]	Testing: 1 /21 university network; 1 /20 university network
Unclean [50]	Training: 47 million public network IPs Testing: Same as training
FluXOR [21]	Training: 50 SPAM domains and normal mails Testing: Same as training
Tight [51]	Training: Crawdad Wi-Fi data Testing: Same dataset
Tamed [52]	Testing: two university /16 nets
Incremental [53]	Testing: internal network capture
Markov [54]	Training and testing: 1 month of 11 C-class university nets
Synchronize [29]	Testing: one host port scanning, two hosts (7-h normal usage)

TABLE 2.4: Comparison of normal sources for detection.

Papers	Botnet Packets
BotMiner [47]	Testing: three compiled virtual IRC (four clients each); one IRC real log (259 clients); two HTTP compiled virtual (four and one client); two P2P virtual (82 and 13 clients)
BotSniffer [45]	Testing: one honeynet-captured IRC bot; three compiled virtual IRC (five clients and two with four clients); two compiled virtual HTTP (four clients and one client); two real IRC text logs
BotHunter [32]	Testing: 10 IRC bots in a virtual network; honeynet: 26 hosts; 2019 infections in 3 weeks; unknown amount of bots
Appendix B [45]	Testing: four compiled virtual HTTP bots (one client each); one compiled virtual HTTP bot (four clients); one compiled virtual HTTP bot (one client)
N-Gram [46]	Training: one IRC botnet from a honeypot (10 clients); one compiled internal virtual IRC bot (60 clients) Testing: one IRC botnet from a honeypot (10 clients); one compiled internal virtual IRC bot (60 clients)
Stability [48]	Training: honeynet P2P storm bot; 13 different versions Testing: Same as training
Models [49]	Training: two third-party IP text report, two public NetFlow logs, bot IPs from IRC logs Testing: bot IPs (unknown source)
FluXOR [21]	Training: 75 SPAM mail domains Testing: training dataset with cross-validation; domains from Web browsing
Tight [51]	Training: internal virtual network, one bot (74 traces) Testing: same as training
Tamed [52]	Training: four virtual bots (21 traces), three honeynet botnets
Incremental [53]	Testing network captures: three internal virtual bots, four third-party virtual P2P bots
Markov [54]	Training and testing: same as normal
Synchronize [29]	Testing: two virtual bots (21 h); five compiled bots in an 18-host university network (one botnet family)

HTTP: Hypertext Transfer Protocol;IP: Internet Protocol; IRC: Internet Relay Chat; P2P: peer to peer.

TABLE 2.5: Comparison of botnet sources for detection.

should be used, and a validation dataset is highly recommended. Furthermore, few proposals have training and testing datasets with both normal and botnet captures.

These tables show that some proposals use the same datasets for training and testing. This practice should be avoided for statistical reasons [38]. The tables also help to compare the amount of data used on each proposal. Whereas some methods used only one bot binary in a virtual network, others captured more than 15 bot families in several universities.

The overall amount of data sources used during training and testing can be seen in Figures 2.2 and 2.3, where the gray-scale corresponds to each detection method. Two

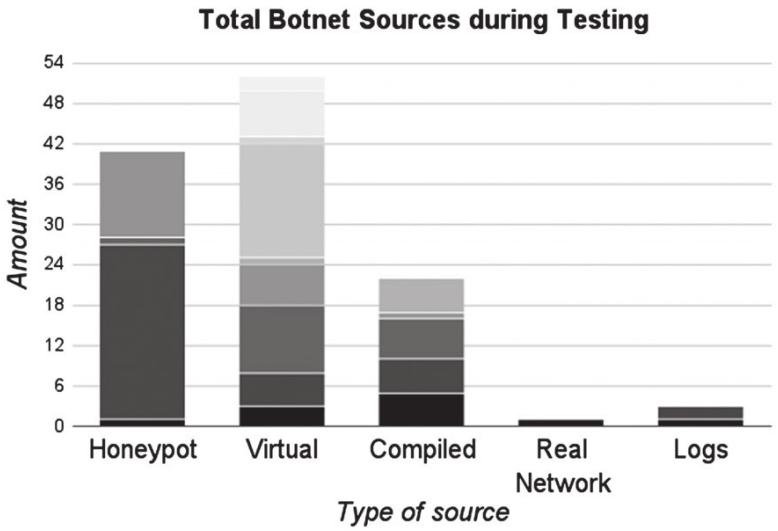

FIGURE 2.2: Sum of botnet sources for all the experiments during testing. Gray scale corresponds to each method.

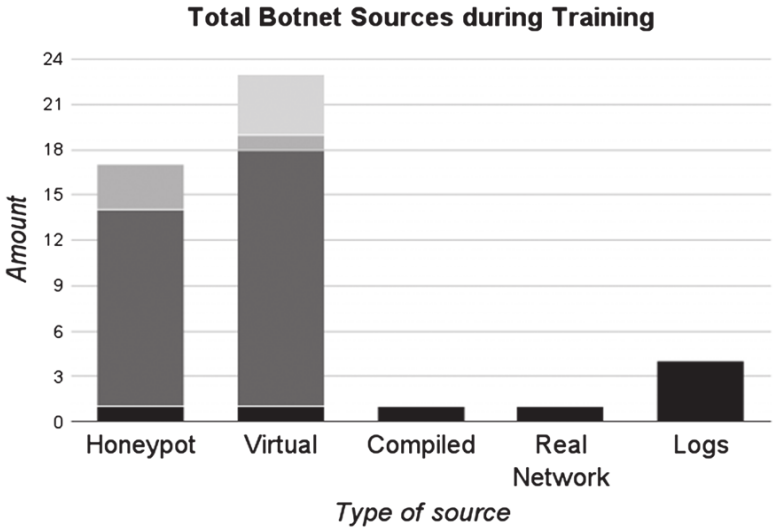

FIGURE 2.3: Sum of botnet sources for all the experiments during training. Gray scale corresponds to each method.

conclusions are presented from the analysis of the graphics. First, most methods prefer controlled environments over real networks to capture botnets. This happens during both the training and testing phases. However, the implications of this decision, such as the type of attacks not captured, have not been estimated. Second, training phases use much less sources than testing phases. This is probably due to the unsupervised methods used.

Papers	Algorithms
BotMiner [47]	X-means and hierarchical clustering (unsup)
BotSniffer [45]	SPRT, threshold random walk (sp), hierarchical clustering (unsup)
BotHunter [32]	-
Appendix B [45]	Autocorrelation (sp)
N-Gram [46]	K-means and X-means clustering (unsup), decision tree (sup)
Stability [48]	-
Models [49]	CUSUM (sp), hierarchical clustering (unsup)
Unclean [50]	-
FluXOR [21]	Naïve Bayes classifier (sup)
Tight [51]	Naïve Bayes, j48 decision trees, Bayesian networks (sup)
Tamed [52]	K-means clustering (unsup)
Incremental [53]	-
Markov [54]	HMM (sup)
Synchronize [29]	EM clustering (unsup)

TABLE 2.6: Comparison of detection algorithms.

2.3.3 Comparison of Detection Algorithms

Table 2.6 shows a comparison of the algorithms and techniques used on each paper. It was rather difficult to create this table because some proposals did not describe the algorithms explicitly. All the papers have used heuristic-based rules at some point of the analysis. The (sup) reference means supervised technique, the (unsup) reference means unsupervised technique and the (sp) reference means signal processing.

2.3.4 Comparison of Accuracy-based Performance Metrics

Accuracy-based performance metrics, as stated in [55], can be measured using multiple techniques. Table 2.7 describes the definition and explanation of the two-class problem metrics. It helps to better understand what the papers reported:

- TP, or true positives, number of times the model predicts positive when the example label is positive.
- FN, or false negatives, are the number of times the model predicts negative when the example label is positive.
- FP, or false positives, are the number of times the model predicts positive when the example label is negative.
- TN, or true negatives, are the number of times the model predicts negative when the example label is negative.

		Predicted	
		+	-
Actual	+	TP	FN
	-	FP	TN

TABLE 2.7: Two-class problem quantities. FN: false negative; FP: false positive; TN: true negative; TP: true positive.

Metrics can be described on the basis of these definitions:

- Percent correct or accuracy is the portion of the test examples that the model predicts correctly: $(TP + TN)/(TP + FN + FP + TN)$.
- Error rate is the portion of the examples in the test set that the model predicts incorrectly: $(FN + FP)/(TP + FN + FP + TN)$.
- Precision is the portion of the test examples predicted as positive that were really positive: $TP/(TP + FP)$.
- True-positive rate (TPR) or recall is the portion of the positive examples that the model predicts correctly: $TP/(TP + FN)$.
- True-negative rate (TNR) is the portion of the negative test examples that the model predicts correctly: $TN/(FP + TN)$.

In [55], it is stated that other researchers have made the case that evaluations that use accuracy metrics are problematic, for they do not show how well a model predicts instances by class. Maloof [55] also stated that "if a testing set contains many more negative examples than positive examples, high accuracy could be due to the model's exceptional performance on the majority". Data in Table 2.8 and the analysis of the training dataset diversity extracted from Tables V and IV should help to evaluate this problem.

Table 2.8 shows that none of the proposals reported all the four basic metrics. If a proposal only reports the TPR value, it is incredibly difficult to understand the significance of its results. Moreover, the table shows that two of the papers did not report any performance metrics at all.

Most of the metrics discussed in [55] were not reported in the papers. In consequence, whenever possible, we computed some of these values ourselves. Our calculations of the missing metrics can be prone to errors, so the labels used in Table 2.8 are shown in the following paragraphs. They help to better distinguish the original values from the calculated values.

Papers	FPR (%)	FNR (%)	TPR (%)	TNR (%)	Percent correct	Error rate (%)	F-Measure (%)
BotMiner [47](A)	0.1875(G)	-	96.82(G)	81.25(D)	-	-	-
BotSniffer [45]	0.16(G)	-	100(G)	84(D)	-	-	-
BotHunter [32]	-	-	(G)	(G)	-	-	-
Appendix B [45](A)	(G)	-	(G)	-	-	-	-
N-Gram [46](A)	8.14(D)	1.64(D)	98.36(G)	91.86(G)	-	-	-
Stability [48]	0(D)	0(D)	100(G)	100(G)	-	-	-
Models [49](B)	-	-	88	-	-	-	-
Unclean [50]	1.21(G)	1.22(D)	98.78(G)	87.9(D)	-	-	-
FluXOR [21]	0	-	-	100(D)	-	-	-
Tight [51]	15.04(G)	2.49(G)	75.1(D)	84.96(D)	-	-	-
Tamed [52](A)	-	6.25(D)	93.75(G)	-	-	-	-
Incremental [53](A)	14.15(G)	0(G)(B)	100(G)	85.85(D)	-	-	-
Markov [54](A)	13(G,B)	11(G,B)	88(G,B)	86(G,B)	87(G,B)	51(G,B)	96(G,B)
Synchronize [29](A)	0.7	3.40	95.86(G)	95.87(G)	95.87(G)	4.1(G)	97.44(G)

TABLE 2.8: Accuracy-based performance metrics comparison. FNR: false-negative rate; FPR: false-positive rate; TNR: true negative rate; TPR: true-positive rate.

References for the computed, non-original values were as follows:

- G: Some metrics were reported for some experiments. However, no final value was reported for all the experiments. If G appears with a percentage number, it means that we could manually compute an averaged total value by closely analyzing the paper (if more than one method was used, we use the best result). The G alone means that we could not deduce a total value from the isolated experiments.
- -: No value or computation was reported.
- A: This paper does not have an automated method to decide whether it detected a botnet or not. Results were obtained by counting the data labels.
- B: Results were reported, but details about the evaluation process were not given. We could not assess if the numbers were correctly computed.

- D: No value was reported for this metric, but we could calculate some approximation as the difference with its counter value. $FPR = 1 - TNR$, $FNR = 1 - TPR$, $TNR = 1 - FPR$ and $TPR = 1 - FNR$.

Table 2.8 is one of the most important outcomes of this Chapter. It clearly shows that some results have not been properly computed. None of the proposals reported all the values. Only one proposal used an alternative method to report results, such as receiver operating characteristic curves [54]. Most of the papers did not compute a total value for each metric (where the G reference was used). Instead, they reported separate values for each experiment. Furthermore, most of the papers did not compute some values at all, forcing us to calculate the missing metrics using its counterpart value (the D reference). Finally, none of the papers reported the percentage correct and the error rate metrics.

2.3.5 Comparison of Unknown Botnet Detection Capabilities

Table 2.9 compares unknown botnet detection capabilities, that is, the ability to detect botnets that were not present in the training dataset. For several reasons, this is perhaps the most difficult desired property to extract: first, because most papers do not describe which botnets were used on each dataset; second, because most of the proposals were not designed to detect unknown botnets and therefore they did not describe this ability; and third, because the unknown botnet concept can be analyzed from different perspectives. An unknown botnet can be considered when the following occur:

- The protocol is unknown.
- The attack is unknown.
- The family is unknown, which implies unknown protocol, C&C servers and attacks.
- The variant is unknown, which implies unknown attacks or unknown C&C servers but known protocol.

These issues made it difficult to decide whether a method could detect unknown botnets. However, this property is crucial to know if other researchers can use the proposal safely.

Table 2.9 shows how likely it is that the method detects an unknown botnet. The analysis of this table reveals that few proposals are prepared to detect unknown botnets. Most papers have strong constraints, for example, only analyzing IRC botnets showing

Papers	HTTP	IRC	P2P	Generic
BotMiner [47]	1	1	1	1[1]
BotSniffer [45]	1	2[2]	1	1
BotHunter [32]	1	2[3]	1	1
Appendix B [45]	1	1	1	1
N-Gram [46]	1	1	1	1
Stability [48]	1	1	2	1
Models [49]	2	2	2	1
Unclean [50]	1	1	1	1
FluXOR [21]	2	2	2	2[4]
Tight [51]	1	2	1	1
Tamed [52]	2	2	1	1
Incremental [53]	1	1	1	1[5]
Markov [54]	2	2	2	2[6]
Synchronize [29]	2	2	2	2[7]

TABLE 2.9: Unknown botnet detection comparison.

suspicious behavior. It is worth noting that two proposals seem capable of detecting unseen botnets almost without constraints [47, 49]. However, none of them have carried out experiments to confirm this capability. The references for Table 2.9 are as follows: 1 means there are no chances of detecting this type of unknown botnet, 2 means there are good chances of detecting this type of unknown botnet and 3 means there is a high probability of detecting this type of unknown botnet. The conditions that should be fulfilled to detect unknown botnets on each paper are explained in the footnotes.

2.3.6 Comparison of Protocol-dependent Features

Most of the proposals base their detection methods on the recognizance of statistical protocol features. Most of the times, this is not a bad practice. However, depending too much on it could lead to a rigid method. Usually, the proposals do not explicitly analyze this dependence. Also, they tend to assume that their statistical features can find all the traffic they are trying to capture. For example, several proposals detect the IRC protocol by capturing only the traffic using the Transmission Control Protocol (TCP) port 6667. However, it is not uncommon to find IRC servers working on other ports as well.

Table 2.10 shows which protocols are statically detected on the proposals. Most of them use the TCP or User Datagram Protocol (UDP) port numbers to filter the protocols. The IDS column identifies the use of statical IDS rules. The operating system column identifies the detection of the operating system with statical rules. The analysis of Table 2.10 shows that several proposals detect botnets using only the TCP. However, botnets have been reported to use UDP-based protocols as well. The exact details about

Papers	HTTP	DNS	SMTP	TCP	ICMP	UDP	IRC	IDS	P2P	OS
BotMiner [47]	-	✓	-	-	-	-	-	-	-	-
BotSniffer [45]	✓	✓	✓	✓	-	-	-	-	-	-
BotHunter [32]	-	-	-	-	-	-	✓	✓	-	-
Appendix B [45]	✓	-	-	-	-	-	-	-	-	-
N-Gram [46]	✓	✓	✓	✓	✓	✓	✓	-	✓	-
Stability [48]	✓	✓	-	✓	-	✓	-	-	✓	-
Models [49]	✓	-	✓	-	-	-	-	-	-	-
Unclean [50]	-	-	-	-	-	-	-	-	-	-
FluXOR [21]	-	-	-	-	-	-	-	-	-	-
Tight [51]	-	-	-	✓	-	-	✓	-	-	-
Tamed [52]	-	-	-	✓	-	✓	-	-	-	✓
Incremental [53]	-	-	-	✓	✓	✓	-	-	-	-
Markov [54]	-	-	-	✓	-	-	-	-	-	-
Synchronize [29]	-	-	-	✓	-	-	-	-	-	-

TABLE 2.10: Comparison of protocol-dependent features.

these statical filters are explained on each proposal subsection in Section 2.4. This table helps us understand how the proposals were designed and also know how to avoid being detected by such methods.

2.3.7 Comparison of the Diversity of the Datasets

This category refers to the various datasets that were captured and the types of botnets analyzed. It is important to have a diverse dataset because every botnet has unique characteristics. The applicability of each proposed method could be analyzed by studying which botnet was included in the training and testing datasets. Tables 2.4 and 2.5 shows that several proposals only used one or two botnets for training and testing. Others used the same dataset for training and testing. This last practice should be avoided for statistical reasons [52].

In the next section, each method is described in depth.

2.4 Detailed Analysis of Previous Botnet Detection Proposals

Previous sections described a topology map of detection characteristics, a set of desired properties of botnet detection papers and a set of tables to compare the detection proposals. This section analyzes the context on which each proposal was created. It also summarizes the steps of each detection method and highlights each proposal

Papers	Verify	Preprocess	Compare
BotMiner [47]	2	2	1
BotSniffer [45]	2	2	1
BotHunter [32]	2	1	1
Appendix B [45]	2	2	1
N-Gram [46]	1	2	1
Stability [48]	2	3	1
Models [49]	2	3	1
Unclean [50]	1	2	1
FluXOR [21]	3	2	1
Tight [51]	2	3	1
Tamed [52]	3	2	1
Incremental [53]	2	3	1
Markov [54]	2	3	1
Synchronize [29]	3	3	1

TABLE 2.11: Generic method comparison.

difficulty, bias, assumption, hypothesis, advance, contribution and result. Each paper is described and analyzed in the next Subsections.

Table 2.11 shows a generic comparison of all the proposals before their detailed analysis. It compares three properties: *verify*, *preprocess* and *compare*. Verify refers to the verification of the datasets, preprocess refers to the preprocessing of the datasets and compare refers to the comparison of the results with other proposals. Each property can be fully developed (number 1), moderately developed (number 2) or not elaborated at all (number 3). Therefore, a quick comparison can be performed before reading the description. The analysis text of each proposal further describes these properties.

2.4.1 BotSniffer

In the BotSniffer [45] paper, three different botnet detection approaches are presented. The first two are analyzed in this subsection and the third one in Section 2.4.2. The first two detection approaches share a common starting base: sniff network packets group together hosts that connect to the same remote server and port and separate these groups in time windows. With these data, the first approach looks for hosts that had triggered some attack fingerprint (such as SPAM sending, binary downloading and port scanning). If more than half of the group had performed attacks, then the group is marked as a possible botnet. The sequential probability ratio testing algorithm is used to decide whether it is a botnet.

The second proposal looks for hosts that answered similar IRC protocol responses using the DICE distance as a similarity function. IRC responses are clustered on the basis of

this similarity measure. If the biggest cluster is more than half the size of the group, then the group is marked as a possible botnet. The sequential probability ratio testing is used again to decide whether the group is really a botnet.

This paper is one of the most cited papers in the botnet detection field. It presents several behavioral techniques to detect botnets that accomplish good results. However, much new data and botnets have been found since its publication.

The dataset used for validation may be too scarce for generalizing the technique. Only one real IRC botnet was captured. The rest of the dataset is composed of one IRC text log and five custom-compiled LAN botnets. The dataset was verified using three methods, but the first two are not effective. The first verification was under the assumption of a clean normal capture because of a well-administered network. However, even well-administered networks can have infected computers. The second was the use of the same BotSniffer method over the dataset. It is commonly not considered a good practice to verify a dataset using the same proposed method. The third verification was performed using the BotHunter [32] proposal. This was a good verification. However, it was only used over the normal captures.

During the preprocessing stage, hard and soft whitelists were used to filter the dataset. Unfortunately, there is no analysis of the bias introduced by these lists. For example, it was found that some non-botnet Web sites made syn- chronic requests that are similar to a botnet request. This possible interference was solved by whitelisting the Web sites. Protocols different than TCP were filtered out.

Performance metrics were computed for FPRs and TPs (not TPR). However, FNR and TN metrics were not computed. The missing metrics could be a consequence of the difficulty to have a labeled dataset. This incompleteness made it difficult to compare the reported results with those of other papers.

The FPR was computed over a dataset that only had normal labels. Under some circumstances, such a metric could be biased. Normally, it is recommended to compute the FPR with a dataset having both normal and botnet labels. Using both labels could be important, as the FPR calculation needs to divide by the total instances analyzed. This issue could be caused by the difficulty of obtaining a labeled dataset. On the other hand, a mixed capture was used to compute the TPR. However, there is no explanation of how was the mixture done. There is also no analysis of the FNR or the TN.

The density-check method is based on the assumption that IRC botnets use very similar messages and that it is unlikely that humans write very similar messages. The results obtained support this idea. The algorithm assumes that the more humans captured in an IRC channel, the more randomness involved. Therefore, it could be less likely to

obtain a homogeneous crowd, that is, more users are less likely to talk similarly than few users. However, in [56], it was stated that *Stability is an important and noteworthy feature of human action (...)*. This may support the idea that more users are more likely to talk similarly. Furthermore, we believe that this assumption of the proposal was the main reason for the 11 FPs in the experiments.

It was also assumed that the TCP would be the primary protocol used for C&C channels. However, it could be important to add the UDP or Internet Control Message Protocol as well. The dataset used was not made public, perhaps because of privacy issues. This made it impossible to reproduce the method. It was not within the goals of the paper to compare botnet attacks and manually automated attacks. However, such a comparison may help to better evaluate the results obtained. The proposal is stated to have implemented a real-time correlation engine, but there was no analysis or further mention. The detection conditions of the proposal need traffic from more than one host to detect botnets.

The paper uses a method called threshold random walk (TRW) within the response crowd activity check algorithm and within the response crowd density check algorithm to decide whether there is strong evidence of a botnet infection. It states that the TRW method needs a large amount of response crowds to decide properly. Although it is true that the response crowd activity check algorithm generates many response crowds, it is also true that the response crowd density check algorithm does not. Unfortunately, this last algorithm was not deeply analyzed in the proposal, and therefore, it may not be suitable for the TRW method. On the other hand, the main detection method was described to be a protocol-independent approach; however, it depends on four conditions: first, on detecting the HTTP; second, on detecting the IRC protocol; third, on detecting the DNS mail exchanger records for SPAM sending; and fourth, on detecting the SMTP.

2.4.2 Appendix B of BotSniffer

Appendix B of [45] proposes to identify HTTP C&C channels by detecting a repeating and regular visiting pattern from one single bot. The proposal uses signal encoding and autocorrelation approaches. HTTP requests are captured, and a time series of two tuples (timestamp and amount of bytes transferred) is generated. The sign of the bytes transferred value represents the packet direction. This time series is analyzed using autocorrelation techniques to find out its time-spatial properties, such as the presence of a periodic signal. Autocorrelation is computed for each lag, and consequently, an autocorrelation series is obtained. Some experiments were conducted to prove

this method, which resulted in several TPs and some FPs. If the proposal would have considered the differentiation between manual (or automated) attacks and botnet attacks, then a more realistic scenario could have been analyzed.

The analysis of this proposal shows that the dataset used for verification purposes is only composed of custom-compiled botnets in a virtual internal network. This dataset might lack real botnets in the wild. The normal dataset used is the same as in the original BotSniffer proposal in Subsection 2.4.1. In this dataset, the BotSniffer program itself was used to test the normality of the BotSniffer validation dataset. It is not correct to use the same method for validation purposes.

Different datasets are used to compute the FP errors and the detection performance. However, in [57], it is stated that *Also, it is important to make sure that the data used for estimating a model and the data used later for testing and applying a model come from the same, unknown, sampling distribution*. Perhaps, different datasets are used because of the difficulty to obtain more trusted data. Also, this might be the cause why the proposal does not compute the FN and TN errors. Moreover, it may be why there are no comparisons with other proposals, and therefore, the results could not be verified. Fortunately, this proposal is capable of detecting botnets by capturing packets in one host.

2.4.3 BotMiner

The BotMiner detection framework [47] has three different phases of analysis. The first phase groups together hosts with similar activity patterns. The second phase groups together hosts that achieve similar attacking patterns, and the third phase groups together hosts on the basis of similarities from the other two phases. The first phase uses flow information such as the Internet Protocol (IP) addresses, ports and the network profile of the flow. It computes statistical measures such as flows per hour, packets per flow, average bytes per packets and average bytes per second. A two-step X-means clustering method is used to create clusters of hosts that behaved similarly in the network. The second phase uses the Snort IDS to extract attacks from each host and to group hosts by attack type. Among the malicious actions detected are SPAM, exploit attempts, port scans and binary download. The third phase computes a score for each attacking host on the basis of previous similarities and uses it to calculate a similarity function between hosts. Finally, a dendrogram is created using this function to find out the best cluster separation. The most important statical characteristics used were the mail exchanger DNS queries to detect SPAM and the 25 TCP ports for SMTP.

The proposal uses both virtualized and real botnet captures. Unfortunately, there is no information about how the botnet dataset was verified. The normal captures were verified using the BotHunter and BotSniffer software. In the preprocessing stage, some well-known Web sites are whitelisted. However, the implications of the filter are not exposed. It also forces the method to maintain a list of well-known hosts, which could be error prone and time-consuming. The results seem encouraging. However, they could not be reproduced because the dataset was not made public. Comparisons with other papers were not carried out. Unlike other proposals, this work uses one novel idea to differentiate between botnets and manual attacks: botnets act maliciously and always communicate in a similar way, but the manual attacks only act maliciously.

2.4.4 BotHunter

The BotHunter framework [32] recognizes the infection and coordination dialogue of botnets by matching a state-based infection sequence model. It captures packets in the network egress position using a modified version of the Snort IDS with two proprietary detection plug-ins. The proposal looks for evidence of botnet life cycle phases to drive a bot dialogue correlation analysis. IDS warnings are tracked over a temporal window and contribute to the infection score of each host. The host is labeled as a bot when certain thresholds are overcome. This proposal associates inbound scans and intrusion alarms with outbound communication patterns.

An infection is reported when one of two conditions is satisfied: first, when an evidence of local host infection is found and evidence of outward bot coordination or attack propagation is found and, second, when at least two distinct signs of outward bot coordination or attack propagation are found.

Five experiments were conducted. The first experiment was carried out on a virtual test bed and used one VMWare Linux, one IRC server and two infected Windows instances. The traffic of this experiment was injected. The second experiment was performed on a honeynet and used nine Windows XP, 14 Windows 2000 and three Linux from a Drone Manager. The third experiment was carried out on a college network and was used for normal packet capture. The fourth experiment used a university campus for normal packet capture, and the fifth experiment was performed in a production network with 130 IP addresses for 10 days. Only one experiment was carried out with normal and botnet data at the same time, probably because of the difficulty to mix two types of datasets together.

Some normal and botnet captures were not completely verified, probably because of the difficulty of performing such a verification. However, the verification of the dataset is the best way to assess whether a variable represents what it is intended to measure [58].

The performance metrics reported in the proposal are incomplete. This might be caused by the limited labels included in each dataset. In the first experiment, a TPR of 100% was reported, but no FPs or TNs were computed. In the second experiment, an FP error of 95.1% and an FN error of 4.9% were reported. In the third experiment, a TPR of 100% was reported. However, no FP, FN or TN were reported. In the fourth experiment, an error of 0.81 FP per day was reported. However, no FN or TP were reported. In the fifth experiment, one FP per 10 days was reported. However, no FN or TN were reported. Finally, the proposal did not compare the results with other papers.

The BotHunter proposal is based on a statical IDS; thus, some of the detections are static. For example, the IRC protocol is identified by means of the 6667 port, and some IP addresses of known botnet servers are embedded into the Snort configuration file. Among the bot statical characteristics detected, the sequence of bytes in the binary download and the Snort statical fingerprints are used by the proposal.

The proposal did not publish the dataset used. This might be due to privacy issues or nondisclosure agreements. There are also no details about the Statistical Scan Anomaly Detection Engine and Statistical Payload Anomaly Detection Engine port scanning detection algorithms. Consequently, the reproduction of the paper might be difficult to accomplish.

The proposal does not seem to be designed to detect bots using only the traffic from one host. Moreover, it is not designed to differentiate between botnets and manual (or automatic) attacks. However, as the model is based on the life cycle of botnets, it is very probable that it could work fine for this situation. The method has two major advances. First, it seems capable of analyzing, detecting and reporting botnets in real time. Second, it is the only proposal that was published as a product.

2.4.5 N-Gram

The N-gram work [46] proposes an unsupervised, classification-based and IRC-based bot traffic detection method. It has two phases: classification of application traffic and detection of bots. The first phase has several stages. The first stage classifies traffic into known applications using signature-based payloads and known ports (signatures were generated previously). The second stage classifies unknown traffic from the previous step using a decision tree based on temporal-frequent characteristics. This tree

differentiates known application communities (the proposal have the temporal-frequent characteristics of known flows). The third stage uses an anomaly-based approach to cluster the traffic. To differentiate malicious traffic from normal traffic, the proposal first creates profiles for known applications; these profiles use a one-gram technique that builds a 256-position vector holding the number of times that every byte appeared in the traffic payload.

The second phase has several stages, and according to the paper, it starts with the feature analysis stage. However, no information is given about this stage, seriously limiting the analysis of the proposal. The real first stage, then, clusters the unknown traffic (which still remains unknown after the first phase) using the K-means, unmerged X-means and merged X-means algorithms. The second stage assigns the label botnet, in the merged X-means algorithm, to the cluster with the lower standard deviation. There is no information about how the labeling is performed in the other methods. The proposal states that the unmerged X-means is the normal X-means and that the merged X-means "tries to find the botnet cluster as cluster with lowest standard deviation and then gather as many instances that represent botnet IRC as possible from remaining clusters."

The proposal assumed that bots reply to commands quickly, that bots communicate synchronously and that botnet IRC content is less diverse than normal IRC. The analysis shows that the training dataset might not be large enough to be considered representative of the problem. It has only one real botnet, one simulated botnet and two normal captures. The paper does not describe any verification of the normal captures.

The proposal is difficult to reproduce, as the labeling process is not described, the performance metrics analysis is incomplete and the dataset was not made public. There is also no comparison between the results presented and other papers. The analysis shows that the method is not completely unsupervised, because the centroids of the k-means algorithm were manually selected. This introduces a bias and makes the method unreproducible. The method presents a novel and simple idea to detect botnet clusters automatically: to search for the cluster with the lowest standard deviation.

2.4.6 Unclean

The Unclean proposal [50] predicts future hostile activity from past network activity. The term spatial uncleanliness is defined as the propensity of bots to be clustered in unclean networks, and the term temporal uncleanliness is the propensity of networks to remain unclean for extended periods. The proposal is divided into two phases.

In the first phase, a dataset is used to evaluate both unclean properties. Although there is no detailed information about the capture procedure, the dataset has two types of information: the first type is external reports of phishing, port scans and SPAM activity, and the second type is some internal network captures. External reports from third parties are used as ground truth for the experiments. The internal capture was performed by observing a public network. The spatial uncleanliness property is evaluated by comparing the population of IP addresses in an unclean report against the normal expected random population of the Internet. If the hypothesis is true, then the IP addresses of the bots should be more densely packed than the randomly selected addresses. The temporal uncleanliness property is evaluated by testing how an old report of unclean addresses predicts compromised IP addresses in the future. It is expected that unclean old reports are better predictors of future IP addresses than randomly chosen IP addresses.

In the second phase, the paper analyzes the performance impact of blocking these future bot IP addresses to differentiate between innocent, unknown and hostile IP addresses. An FP is considered each time an innocent IP address is blocked, and a TP is defined each time an unclean IP address is blocked. Unknown IP addresses are not used, probably adding a bias to the results. Results supported the hypothesis of spatial and temporal uncleanliness in the experiments.

An analysis of the validation experiments for the first phase suggests that the results are incomplete. A TPR of 90% was reported, but no FP, TN, FN, FPR or FNR were reported. Also, results are not compared with those of any other paper. Two assumptions were made: first, that bots are always used to attack and, second, that if a 5-month-old unclean report can predict current unclean activity, then a recent report should be more effective. The analysis of this proposal shows that the verification of the external reports was not described and that the dataset was not published. Also, it was not in the design of the proposal to differentiate between botnets and other types of attacks.

2.4.7 Tight

The Tight proposal [51] aggregates traffic to identify hosts that are likely part of a botnet. The hypothesis is that some evidence of botnet IRC C&C communication activity can be found by monitoring IRC traffic at a core location. The training dataset is composed of 600 GB of wireless normal traffic from the CRAWDAD archive and 74 flows of a compiled botnet from an internal test bed. The proposed method has five steps. First, flows are mixed in a common dataset (unfortunately, there is no information about how the mixture was performed). Second, a whitelist and a blacklist are used to

filter the flows. Third, flows are classified according to their chat-like characteristics. Fourth, correlations are used to find similar C&C flows. Fifth, a topological analysis of the flows is carried out to identify common connection endpoints and to manually determine which is the traffic between the controller and the rendezvous point.

Several assumptions are made in the paper: first, that the IRC-based botnet command messages sent by the Botmaster are brief and text based; second, that two different flows of the same application behave similarly; and, third, that botnet C&C channels do not sustain bulk data transfers. The analysis shows that some filters overfits the data, such as deleting the high-bit-rate flows. However, these filters were not verified, and they seem to be created specifically to filter out the already-known botnet traffic.

There is an issue regarding the dataset. When the botnet packets were obtained, the secure shell [59] traffic generated from the test bed setup was also captured. This biases the capture. Almost every dataset has a bias, but it is important to enumerate and consider them [58]. Furthermore, the normal captures were not verified, and the dataset was not published.

Another bias is present in the results. Some results were obtained by applying the classifiers to one dataset; other results were obtained by applying the classifiers to another different dataset. It is commonly accepted [58] that there should be at least three types of datasets: training, testing and validation. An evaluation methodology should also be used [60].

The design of the proposal is considered to detect in real time, but unfortunately, no experiments were carried out to support this idea. A major highlight of the paper is that it focuses on novel behavioral features: a bot communicates with a C&C server, the server is controlled by a Botmaster and the bots synchronize their actions.

2.4.8 Stability

In Stability [48], P2P botnets are detected using flow aggregation and stability measurements. The hypothesis is that the proposal "can further distinguish the structured-P2P-based bot from normal clients by detecting the stability of I-flow." The dataset was created by capturing background traffic from a university campus and later injecting real botnet traffic into it. The botnet traffic was captured in a honeynet, and thus, it is verified as malicious by the definition of a honeypot.

This proposal is divided into two phases. In the first phase, the flow features are extracted. In the second phase, these features are used to compute the stability of the flows. Before the first phase, the dataset is divided into 1-h time windows.

The first phase is composed of several steps. In the first step, protocol flows are aggregated on the basis of the double two-tuple IP addresses and ports. In the second step, for each aggregated flow, two novel values are computed: flows per aggregation flow (fpf) and average bytes per aggregation flow (abf). In the third step, an entropy value that uses the fpf as a random variable is computed. A high fpf and an abf lower than 300 bytes were used to detect bot flows.

The aim of the second phase is to compute the flow stability with two sliding windows: one serves as a base- line and the other as a detection window. In the first step, the distance between the abf of both windows is compared against a predefined threshold. Every time the distance overcomes the threshold, a flow change is counted. In the second step, windows are moved forward, and the stability index of the flow is computed. If the stability index is above a threshold of 90%, then a botnet is detected.

The proposal assumes that common DNS ports are not going to be used for C&C channels and that the common P2P ports are enough to detect that protocol. The analysis of the proposal shows that the preprocessing stage adds a bias to the results. The first bias is added when DNS traffic is filtered out after discovering that it has a stability index similar to that of P2P botnets. The second bias is added when port 28000 is filtered after discovering that it is being misdetected by the algorithm. Reproduction of the method is not possible because of two reasons. First, the paper uses several unspecified ad-hoc experimental values. Second, the dataset was not made public. Unfortunately, the normal traffic was not validated.

2.4.9 Incremental

In Incremental [53], the botnet activity is monitored on the basis of similarities between the traffic feature streams of several hosts.

The proposal is divided into five steps. The first step captures both normal and botnet traffic from an internal network. P2P botnet traffic is obtained from a third party. The captures in this step are simulated, as the internal network did not have access to the Internet.

The second step reduces the volume of normal data by applying some filters. Non-TCPs are filtered out, several whitelists and blacklists of common Web sites are applied and packets with more than 300 bytes are deleted. The third step creates the feature streams. Traffic is divided into 5-min time windows, and two features are computed: average bytes per packets and packet amount.

The fourth step computes a distance measure between streams using an incremental discrete Fourier transformation technique. The novel characteristic of this technique is that it does not need to recalculate all the coefficients every time a new value arrives.

In the fifth step, the Snort IDS is used to manually verify the activity of the suspect hosts (hosts with high similarity streams). In this step, it is also decided whether this host is part of a botnet or not. Unfortunately, no information about this step is given. The proposal assumes that simulated botnet traffic behaves like real botnet traffic and also that the traffic from their network is normal.

The proposal was not designed to differentiate between botnet and other attacks. The analysis shows that the results are unclear. No FNs were computed, and a perfect detection accuracy is reported. The paper also states that the "false positive rate is still relative low because we will analyze their activities to confirm the final result." Unfortunately, the confirmation is not in the proposal. The results are not compared with those of other papers. The proposal uses several filters in the preprocessing step. However, it does not seem that the bias added by these filters would have been evaluated.

Different distance measures are used in the proposal. However, there is no information about the distances called protocol distance measure and start time distance measures. Consequently, it is difficult to reproduce the method.

During the validation experiments, normal traffic and botnet traffic were mixed. Unfortunately, the mixture process is not explained.

This proposal has two highlights. First, it seems capable of monitoring botnets in real time using the time series-like feature streams. Second, the feature stream implementation, although not novel, is an important improvement in the area.

2.4.10 Models

The Models [49] proposal identifies single infected machines using previously generated detection models. These models are composed of two parts. The first part extracts the strings received by the bot on the network to find the commands. The second part tries to find the responses (attacks) to the command sent. Models are stored for later use in real networks. The novel idea is to search for command signatures (string tokens) within the network traffic and then search for the responses to these commands. This technique correctly separates botnets from other attacks. The proposal is separated into five phases.

The first phase consists of several steps. In the first step, traffic from 50 bots is captured. In the second step, bot responses are located within the traffic. This is carried out by separating the traffic into 50 seconds time windows and by computing eight features for each time window. The features are the number of packets, cumulative size of packets in bytes, number of unique IP addresses contacted, number of unique ports contacted, number of non-ASCII bytes in the payload, number of UDP packets, number of HTTP packets and number of SMTP packets. In the third step, the features are normalized. In the fourth step, the change point detection algorithm uses these features to detect where the traffic changes. This algorithm uses the cumulative sum method to detect the changes.

Once the change point is detected, in the fourth step, data are extracted from the traffic to create the profiles. Bot command profiles are composed of string snippets extracted from the 90-s time windows where the change point detection took place: the number of UDP packets, number of HTTP packets and number of SMTP packets and IP addresses. The final response profile is the average of these vectors over the total time of the bot response.

In the second phase, models for commands and responses are created. A model has two parts: the strings for command detection and a network-level description for response detection.

In the first step of the command model creation, snippets that contain the same tokens are grouped together using the longest common subsequence algorithm. In the second step, response traffic payloads are clustered together using hierarchical clustering. In the third step, snippets from the first-step clusters are grouped according to the second-step traffic clusters (snippets likely to generate the same response). In the fourth step, the precision of the method is improved by deleting the tokens found in previously generated and unverified normal captures.

Still, in the second phase, in the first step of the response model creation, the element-wise average of individual profiles is computed on each cluster. In the second step, some FPs are filtered out using a threshold. If none of the following thresholds are overcome in 50 s, then the response is discarded: 1000 UDP packets, 100 HTTP packets, 10 SMTP packets and 20 different IP addresses.

Once the models are created, the proposal implements them. In the third phase, previous models are mapped into the Bro IDS. Bro is configured to work in two stages. In the first stage, if a Bro command signature is matched, then Bro changes to stage 2. In stage 2, a bot detection is achieved if any of the following traffic thresholds are

overloaded: the number of UDP packets, number of HTTP packets, number of SMTP packets or number of unique IP addresses.

In the fourth phase, an evaluation is performed to obtain the performance metrics. A total of 416 binary bots (18 bot families) were obtained from the Anubis service along with 30 non-described storm captures. The evaluation process is performed in several steps.

In the first step, detection models are created using a training set composed of 25% of the total traces. In the second step, results are obtained using a testing set composed of 75% of the total traces.

In the fifth phase, two real-world experiments are carried out. First, the already configured Bro IDS is used in a university campus network, and later, the same Bro IDS is used in an organizational network. The HTTP is detected by filtering the TCP port 80, and the SMTP is detected by filtering the TCP port 25.

The analysis shows that the bot families are separated by hand while creating the models, probably because, once created, the models are useful for a long time. On the other hand, the FPs are manually verified. These issues make the verification of the method difficult.

The proposal seems to use a good dataset, but the information included is not enough to verify its diversity. Also, the third-party Anubis dataset is not verified. Moreover, there is no information about the preprocessing of the dataset.

A highlight of this proposal is that the results are compared with those of the BotHunter proposal using the Models proposal [49] dataset.

2.4.11 FluXOR

The FluXOR proposal [21] detects and monitors fast-flux-enabled domains. Given a fully qualified domain name, its goal is to verify whether it concealed a fast-flux service network and, in such a case, to identify all the agents that are part of the network. Although not strictly a botnet detection technique, it is one of the few proposals that could detect fast-flux botnet domains. Suspicious domains are detected when traces of fast-lux characteristics are found in the DNS and WHOIS information of the domain. A fast-flux behavior is detected with nine features that describe the properties of the domain, such as the degree of availability and the heterogeneity of the hosts in the DNS A records. The experiment setup includes a collector module, which harvests domains from different sources; a monitor module, which monitors domains; and a

detector module, which feeds the classifier algorithm from the Waikato Environment for Knowledge Analysis suite [60]. This proposal does not have a detailed performance metrics analysis, so it is not possible to evaluate how well it performed. A 0% FP was reported; however, no other results were given.

The analysis shows that the malware domains are extracted from SPAM mails and normal domains are extracted from Web history. However, there is no description about the verification of the domains. Unfortunately, the dataset was not made public, probably because of the privacy and security issues related to the domain names. On the other hand, it may be possible to reproduce the algorithm, as the Waikato Environment for Knowledge Analysis framework is publicly available [60].

The design of the proposal does not include real-time detection capabilities. However, under some conditions, it could be possible to detect fast-flux botnets within a very short time.

2.4.12 Tamed

In the Tamed proposal [52] it is hypothesized that ”even stealthy, previously unseen malware is likely to exhibit communication that is detectable. The proposal aims to ”identify infected internal hosts by finding communication aggregates, which consist of flows that share common network characteristics.” This is the first proposal that does not try to detect botnets themselves but presents a coherent and useful set of features in which future work can rely their detection methods on. The work is divided into two phases: the definition of their aggregate feature function and the evaluation of experiments.

In the first phase, three different aggregation functions are defined: destination aggregation function, payload aggregation function and common platform function. The destination aggregate function finds suspicious destination subnetworks for which there are a large number of interactions with the internal network (and the IP addresses involved). This is performed by comparing a baseline past network record with the current network record. The function is composed of two steps. First, the past normal traffic is used to remove the periodic communications. Second, the principal component analysis algorithm is used to find the most important components, and then, a clustering technique is used to find the hosts that connect to the same combinations of destinations.

The payload aggregate function uses the edit distance with substring moves to output a normalized ratio. This ratio indicates how many distances are below a given threshold

between two payloads (and thus need training). The proposal also contributes an algorithm for approximating the fraction of relevant record pairs that satisfy this distance (using a variant of the k-nearest neighbor).

The *common platform* function uses two heuristics for fingerprinting host operating systems passively: time-to-live fields and communication characteristics (such as Windows computers connecting to the Microsoft Time Server). This last function returns the largest fraction of internal hosts that can be identified with the same operating system. The proposal assumed that the C&C server can be in a different network and that bots will not use the Tor network. The analysis shows that the normality of the traffic captured at the university is not verified but granted as normal.

In one of the experiments, the traffic from one botnet was merged with traffic from the university. The IP addresses of the bots were substituted with the IP addresses of hosts in the university. The intention was to have IP addresses with normal behavior patterns and botnet behavior patterns at the same time. However, no analysis was performed about how this change could affect the common platform aggregate function. This issue might be the reason of some FNs during one experiment. Unfortunately, the dataset was not made public, so it is difficult to reproduce the method.

The proposal stated that a 100% TPR was achieved with the exception of one experiment. In this isolated exceptional experiment, an 87.5% TPR was reported. The results of all the experiments should be computed together to calculate the performance metrics. There cannot be any experiment exceptions while computing the results. The proposal was not designed to detect bots using only the traffic from one host. Also, it was not designed to differentiate between botnets and other types of attacks.

2.4.13 Markov

Port scan activities of botnets are detected in [54]. The scanning phases are represented with text symbols, and a hidden Markov model (HMM) is used for training and detection. The work is divided into four phases. In the first phase, a system is used to capture information from the network and generate network logs. Unfortunately, this system is not described. Data were captured for 1 month in 11 C-class university networks. Incidentally, an unknown amount of hosts was found to be infected.

In the second phase, the amount of TCP packets present in certain time windows with SYN and ACK bits is computed. Unfortunately, no information about the time windows is given. The amount of packets is plotted, and each distinct shape of traffic is assigned

a different text symbol. This phase supposedly ends up with a string of letters (one gram) that represent each port scan (observations).

In the third phase, the training phase, the Baum–Welch algorithm is used for 6 h to find a Markov model that maximizes the probability of each sequence of observations.

In the fourth phase, strings are extracted from the rest of the traffic (supposedly of unknown type). However, there is not much information about this. Then, the Viterbi algorithm is used to find the most probable sequence of states in an HMM from each group of observed states, that is, to find the model that better explains the observations. The result of the Viterbi algorithms is used as a type of score. We suppose that they selected the sequence of stored port scan strings that maximizes the score. At some point, a comparison is made between a botnet pattern and an unknown pattern using a similarity function. Finally, the early detection of port scans is defined by these similarity values. However, no information about any threshold or decision boundary is given. The proposal assumed that the botnets only scan TCP ports.

The analysis shows that the dataset used is not verified. There is no clear description of the dataset. Moreover, it is stated that the dataset contains both unknown data and botnet data, but it is not labeled. Also, results were not compared with those of other papers.

Unfortunately, it is difficult to completely understand the method. There is no information about how the port scan detection is performed. Also, the proposal states that the method behaves fine with Monte Carlo simulations, but there is no information about this. The experimental setup is not described.Furthermore, a bigger detection system is described, with botnet policy management and classifications steps, but there is no information or explanation about it. The information given about the automatic detection of botnets does not have enough details.

2.4.14 Synchronize

The characteristic used in [29] to detect bots is network synchronization and the technique used is clustering. There are five phases. In the first phase, data are captured. Three different test beds were used to capture five datasets. Three of the datasets correspond to botnet captures and two to non-botnet captures. The non-botnet captures include a manual port scan and normal traffic. Verification of the captures is carried out for both botnet and non-botnet datasets. The botnet binaries are verified with the VirusTotal Web service [61] and the EUREKA! automated Malware Binary Analysis Service [62]. The botnet traffic is manually verified by experts. The non-botnet traffic

is verified by a fresh installation of the operating system, antivirus programs and the Snort IDS. In the second phase, the TCP flows are extracted using the tcptrace tool.

In the third phase, flows are divided into one second time windows, and three features are used to aggregate them: the amount of unique source IP addresses in a time window, the amount of unique destination IP addresses in a time window and the amount of unique destination ports in a time window. Each instance represents a time window with the aggregated features.

In the fourth phase, the expectation–maximization algorithm is used to cluster the instances. A dataset with both types of traffic was obtained by merging botnet and non-botnet data. Four experiments were conducted: first, to separate between botnets and port scanning activities; second, to separate between botnet and non-botnet traffic from one host; third, to separate between botnet and non-botnet traffic in an unbalanced dataset; and fourth, to separate between a bot and a network of non-botnet hosts.

The fifth phase analyzes the results. The proposal considers the usual definition of FPs and FNs. However, it also includes a novel definition of error metrics based on the administrator perspective, that is, to compute an FP only when a non-botnet IP address is detected as a botnet at least once during a period and also to compute an FN when a botnet IP address is always detected as a non-botnet in a time window.

The proposal assumes that botnets tend to generate new flows almost constantly and that botnets' most typical characteristics are maliciousness, being remotely managed and synchronization and also that botnets only use the TCP protocol.

The analysis shows that the dataset included non-botnet data and manual attacks. However, it is rather small, and it does not cover a large proportion of botnet behaviors. Only five captures were made, and none of them was made in a large network.

Unfortunately, there is no justification of the one second time window used. The results were not compared with those of other papers. One of the highlights is that all the tools and datasets used were made public on the Internet. These datasets might allow other researchers to verify the method. Another highlight is the differentiation between botnet and port scanning traffic.

Unfortunately, no automatic botnet detection method was established, meaning that the final decision is left to experts.

The next section discusses and analyzes the main issues found in the area.

2.5 Discussions About the Network-based Botnet Detection Area

This section discusses the issues found while analyzing and comparing the previous proposals of botnet detection. The analysis showed which problems were solved, which algorithms were used and which questions remained to be answered. We conclude that the botnet detection area has obtained good results. However, it still has many issues to tackle. The following paragraphs discuss some general issues, and the next subsections discuss four groups of common problems.

Reproducibility Unfortunately, most methods cannot be reproduced. We believe that there are two main factors that prevent it. First, the datasets were not published. All the papers except [29] did not publish their datasets. Second, there was not enough information about the methods. Without the step-by-step description of the algorithms, the threshold used and the whitelists, reproduction might be unachievable. The following proposals did not publish all the necessary information to reproduce their results: [48][45][47][32][46][50][51][52][53][54].

Amount of hosts The amount of hosts needed in the network to detect botnets should be considered. It normally depends on the design of the proposal. Most proposals need a large amount of hosts and packets to obtain meaningful results. This limitation might play a negative role in the future adoption of the method. Such amount of hosts can only be obtained by capturing packets on large networks. If the organization that uses the method has such a network, then access to the network can be also made difficult by the nondisclosure agreements that must be fulfilled. On the other hand, if the organization does not have such a network, then it can be extremely difficult to have access to the data. Therefore, some techniques might be only valuable for Internet service providers, large organizations or universities. Currently, there are only two proposals that detect bots using the traffic of one host alone: Appendix B of [45] and [29].

Features An important phase of a botnet detection research is the preprocess of data before the feature extraction [58]. Some proposals tend to overfilter the data and to produce algorithms that work better with a specific dataset. These decisions should not add a bias to the experiments. A data dredging bias can occur when researchers either do not form a hypothesis in advance or narrow the data used to reduce the probability of the sample refuting a specific hypothesis. An example is the whitelist of known Web sites under the assumption that there is a low probability that they will be part

of a botnet in the future [45, 47, 53]. This decision might be taken because most of the traffic on the datasets was directed to popular Web sites. However, the assumption that these sites can be safely whitelisted should be verified, as these sites were successfully attacked in the past, such as Google and Adobe. Furthermore, another source of bias was the assumptions made to filter out the packets that did not fit the method. For example, some proposals filter out packets of the UDP or Internet Control Message Protocol without further explanations [29, 45, 53], narrowing the sample to match their hypothesis. In general, this type of decision is not recommended, as some bots use the UDP exclusively [63].

Experiments Performance metrics are important in every experiment and therefore should be correctly computed [60]. However, some experiments were incorrectly created. For example, some proposals compute the TPR and FNR metrics using only a botnet capture [32][49][50]. Other proposals compute the TNR and FPR using only a normal capture [45, 49, 50]. These types of metrics do not help us understand the results, in the former case, because always predicting a positive outcome generates a 100% TPR and a 0% FNR and, in the latter case, because always predicting a negative outcome generates a 100% TNR and a 0% FPR. More interesting results would be obtained if both normal and botnet captures were used in the experiments.

Metrics Two important issues were found regarding the accuracy-based performance metrics. The first issue, shown in Table 2.8, is that most proposals did not compute the entire set of metrics. This means that most proposals cannot be compared. It is difficult to understand how well the methods performed. The second issue is that some proposals did not clearly describe the design of its experiments. For example, in some papers, it was impossible to tell if they detected 90% of botnet IP addresses, 90% of botnet flows or 90% of botnet actions. There is a huge difference between detecting 90% of botnet IP addresses within a 5-min time window and detecting 90% of botnet flows within a 5-min time window. In the next subsections, the discussion is extended to more specific topics.

2.5.1 Issues Related to the Comparison of methods

There are few works that compare different detection methods using a common dataset. Some of the main reasons for this are that the datasets tend to be private and the description of the methods tend to be incomplete. Also Tavallaee et al. [64] discussed the difficulty of reproducing a method because there is an absence of documentation of the methods and experiments.

A proposal that made a comparison with a third-party method was presented by Wurzinger et al. [49]. This paper identifies single infected machines using previously generated detection models. It first extracts the characters strings from the network to find the commands sent by the C&C and then it finds the bot responses to those commands. The authors downloaded and executed the BotHunter program of Gu et al. [32] on their dataset and made a comparison. The paper compares the results of both proposals and only reports the TPR (True Positive Rate) error metric and the FP values. However, reporting only these values is not enough to know how well the method performed because there is no information about the FPR, the balanced F-Measure or the Error Metric.

Another work that was compared with a third-party method was presented by Zhao et al. [65]. This proposal selects a set of attributes from the network flows and then applies a Bayes Network algorithm and a Decision Tree algorithm to classify malicious and non-malicious traffic. The third-party method used for comparison was again BotHunter. There is a description of how BotHunter was executed, but unfortunately the only error metric reported was a zero False Positive.

The last proposal that also compared its results with a third-party method was made by Li et al. [66]. They analyze the probable bias that the selection of ground-truth labels might have on the accuracy reported for malware clustering techniques. The paper states that common methods for determining the ground truth of labels may bias the dataset toward easy-to-cluster instances. This work is important because it successfully compared its results with the work of Bayer et al. [67]. The comparison was done with the help of Bayer et al., who run the algorithms described in Li et al. [66] on their private dataset.

2.5.2 Issues Related to the Datasets

The importance of using a good dataset, as stated in [51], is sometimes underestimated. Rossow et al. [9] presented a good paper about the prudent practices for designing malware experiments. They defined a prudent experiment as one being correct, realistic, transparent and that do not harm others. After analyzing 36 papers they conclude that most of them had shortcomings in one or more of these areas. Most importantly, they conclude that only a minority of papers included real-world traffic datasets in their evaluations. During the analysis of the papers in this chapter we found several issues regarding the use of datasets.

The first issue is that proposals tend to overlook the verification of the datasets. A dataset in [54] was used for training and validation without verification. Furthermore,

a proposal invalidated its own dataset by including in their capture the secure shell traffic used to set up the virtual environment [51].

Second, the amount of botnets captured is not evaluated. Some proposals [46, 51] captured a very small amount of botnets. If the proposal tried to detect HTTP botnets, then a large amount of different HTTP botnets should be captured for the sample to be representative of the population.

Third, the amount of time spent on the capture is not evaluated. The complete behavior of botnets cannot be captured in a few hours. However, most proposals tend to capture botnets during a short time. Depending on the design of the proposal, this could lead to an incomplete view of the botnet behavior. It is recommended to capture traffic during long periods.

Fourth, labels are not verified. Some proposals captured traffic from an internal network and labeled it as normal without verification [32, 45, 48, 51–54]. However, it is common to find infected computers inside real networks. Without verification, this type of traffic should be considered background instead. Botnet captures are usually labeled as malicious only because a malware binary was used, but no verification is carried out. There is no definition of what should be considered as botnet traffic. Should the traffic generated by a malware process or the one generated by an infected computer be considered as a botnet? Because an infected computer also generates normal traffic, the answer is not straightforward.

Fifth, botnets are not distinguished from other types of traffic besides normal. However, it could be helpful to distinguish botnets from manual and automated attacks. If a proposal uses common attacks to detect botnets, care should be taken not to have FPs. Only one proposal was designed to use the botnet attack responses and signatures to better differentiate botnets from attacks [49]. Another consideration should be made with port scanning activities. Should it be considered part of a botnet or not? Port scanning is commonly considered an attack, but it is also a well-known administrative tool in most networks. Only one proposal differentiated between port scanning and botnets [29].

Sixth, there is a need for a public botnet dataset. This is perhaps the most important conclusion of this chapter. It had been stated that big data should be available for research and validation [68]. Without a public dataset, it is not possible to compare methods. Currently, proposals are forced to create their own datasets. The following reasons contribute to the lack of a common and public dataset. First, proposals use different botnet features for their analysis. Therefore, each one needs to create an appropriate dataset. Second and most important, traffic captures have much private

information that is dangerous to publish. Researchers usually had to sign nondisclosure agreements with the network owner before having access to the data. Furthermore, these captures usually include malware information that should be kept private to avoid spreading the malware. The importance of a public dataset is reinforced by these limitations. A public and complete dataset could be a very valuable contribution to the area. To be useful, it should be correctly designed, it should include every type of botnet, it should be regularly updated and, most importantly, it should be correctly labeled. The creation of a common dataset is perhaps one of the most important issues that the area has yet to achieve.

Seventh, there is an issue in the representation of network traffic. It is common to need huge amounts of disk storage early in the capture process. Researchers have to solve the problem of how to store useful data long enough. The common solution is to use network flow description methods. The idea is to reduce the volume of unused data and retain as much useful data as possible. Two standards are a common choice: IP Flow Information Export (IPFIX) and Argus network flows. The IPFIX standard was defined by the Internet Engineers Task Force, and it is based on the original NetFlow standard. Argus uses its own network flow definition for the same task. One difference between Argus and IPFIX is that Argus flows are bidirectional, whereas IPFIX flows are unidirectional. This subtle difference can have a critical impact on results and should be evaluated.

Eighth, the issue about how to obtain good botnet binaries. The most common options are as follows:

1. Use your own honeypots to obtain the binaries.
2. Obtain binaries from third parties, such as antivirus or other institution honeypots.
3. Obtain and compile the source code from the Internet.

The first choice is the best, but it can take a long time to obtain useful binaries. The second option is the most common. The third choice is good if the test bed has a connection to the Internet and no services are simulated.

2.5.3 Issues Related to the Experiments

To achieve meaningful results, it is important to carefully design the experiments. Some papers correctly proposed to mix normal captures with botnet captures to obtain a better dataset. However, the mixture processes are not often explained [32, 45, 51, 53]. To mix a

dataset means to decide the balance of the mixture. Different balances produce different results and would have different benefits. In [55], it was advised that an unbalanced dataset can bias the results. Nevertheless, it is common to have more packets of normal activity than botnet activity. The mixture of network packets is not easy. There are two main options: to concatenate the traffic or to merge it. To concatenate is easier but more unreal. To merge is better but more difficult to obtain.

An uncommon solution to the merging issue is to change the IP address of one botnet host for the IP address of one normal host [52]. The solution tries to have an IP address with both types of behavior. However, this change could have unpredicted implications in the performance metrics. In fact, in [52], a problem is detected on the passive operating system detector results.

2.5.4 Issues Related to the Detection Methods

The botnet behavior continuously changes, and detection methods should change also. Supervised methods could detect known botnets, but new botnets are still difficult to detect. Methods should adapt to the changing behavior of botnets and should be more flexible. There is a need to increase the detection of unknown botnets.

Several proposals used unsupervised methods [29] [45] [47] [46] [49] [52]. However, none of them verified the methods with unknown botnets.

2.6 Conclusions

This Chapter reviewed and compared the previous surveys on botnets and the previous botnet detection proposals. From the comparison of previous surveys we conclude that there was no common comparison criteria of the detection proposals, they had undefined terminology, they focused on different aspects of botnets, they did a narrow analysis of the papers and they covered a small number of papers.

The previous detection proposals were compared from three points of view. First, as a topology of the *algorithms*, *techniques* and *sources*. Second, as a comparison of the 20 most desired characteristics of any detection proposal. Third, as a deep description of each step in the detection methods. All these comparisons helped us find the issues in the botnet detection area that still need to be solved.

During this Chapter we encountered two limitations. First, some proposals did not publish the details of their algorithms and therefore it was impossible to know what

they did. Second, it was not possible to access to most of the datasets used by the proposals. These problems highlight the difficulty to compare the papers.

The issues detected in the botnet detection area were, the:

- use of unverified captures for training or validation purposes,
- lack of public datasets,
- small amount of botnets in the datasets,
- wrong mixing of packets to create datasets,
- inaccurate outcomes of experiments,
- whitelist of common Web sites,
- lack of comparison with other proposals,
- inaccurate report of performance metrics and
- common overfitting of the preprocessing methods.

Among the reasons for these issues are the fast evolution of botnets and the difficulties in obtaining real and working botnet traffic [9].

The contributions of this chapter can be summarized as follows:

- A novel comparison and summary of previous surveys in the area.
- A novel comparison and classification of botnet network detection characteristics and properties.
- An analysis of the desired properties of botnet detection papers.
- An analysis and discussion of the most important proposals.
- A discussion of open problems in the botnet detection area.

Considering these conclusions and the data analyzed, a final word can be said about the botnet detection methods. The intention is not to find the best methods but to point out the probably most useful detection approaches. The complexity of the botnet behavior seems to be better detected with time-based dynamic approaches, that is, approaches that adapt to changes over time. To detect unseen botnets, it seems to be a good path to use the most general features, although some particular behaviors could be missed. It also seems to be a good approach to use hybrid detection methods, where a meta-learner

can weigh and decide over the results of different behavior detection algorithms. The results obtained by these types of approaches on IDSs are encouraging [69]. Finally, if several algorithms are used in parallel along with a huge amount of data, then, it may be useful to implement big-data solutions [70]. We conclude that the area has accomplished great researches, tools and results, but it still has open problems to be solved.

Chapter 3

The SimDetect Method: Detecting the Similarities Between Bots

3.1 Introduction

In this Chapter we describe the SimDetect method, our first botnet detection method. It was created to tackle a problem that it is believed to be simpler than detecting complete botnets in the network: detecting a unique infected computer alone. This problem may is simpler because there is considerably less traffic, the actions are not so different from each other. The development of this method help us to get familiar with the bot behaviors and to test a simple methodology on real traffic.

The SimDetect method works under the assumption that the most typical characteristics of botnets are *maliciousness* (attacking and infecting, sending SPAM, DDoS, etc.), being *remotely managed* and *synchronization*. The synchronization is not only between different bots, but also within the same bot. For example, the botnet module that sends SPAM and the module to maintain the C&C channel may stop sending packets at the same time when the bot receives the order of not sending any more mails. In this case the same single bot has synchronized two different network connections. The SimDetect method focuses on this type of synchronization.

The basic idea of the method is to first separate the traffic in time windows, then to compute some features for each time window and then use a clustering algorithm [71] to group the processed time windows. The features computed for each time window are the amount of unique source IP address (referred hereafter as *sips*), the amount of unique destination IP address (referred hereafter as *dips*) and the amount of unique

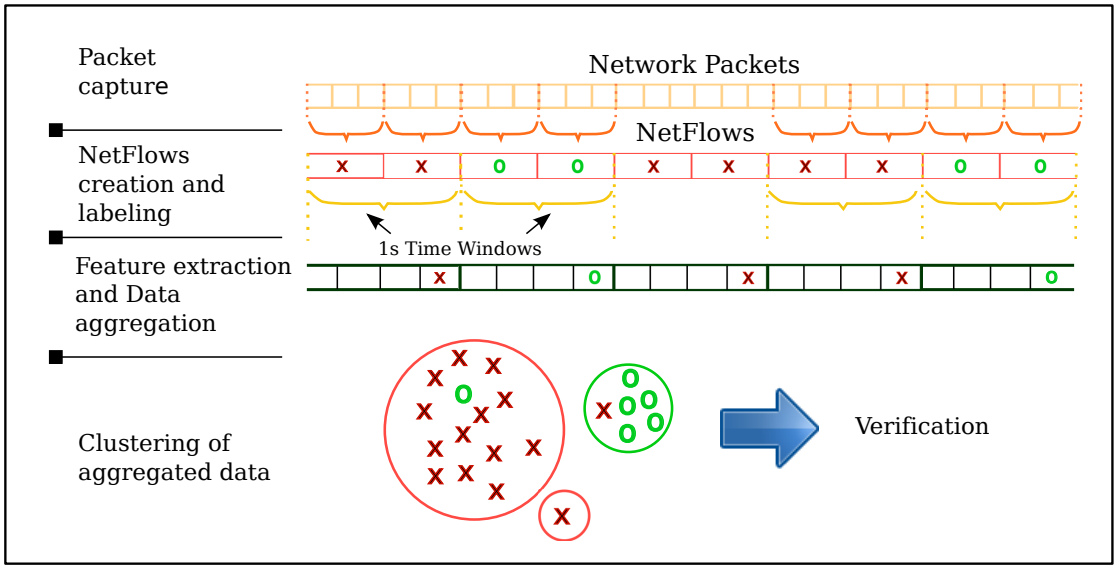

FIGURE 3.1: Summary schema of the SimDetect method

destination ports (referred hereafter as *dports*). The three selected features had been used to detect manual network attacks before [72].

The selected features may reveal the relationships between the number of IP addresses and ports among time intervals, which may characterize the most inherent botnet activities. It is known that during some botnet life cycle phases, a single bot computer generates high network flow rates within very short time periods [45], for example, during Spam sending, DDoS attacks, network scanning and botnet distribution. Our first hypothesis is that these features can also be used to identify the network behavior of bots.

Since this was our first proposed detection method we had to create a botnet dataset from scratch. The dataset consisted in normal traffic from a university, botnet traffic and port scans. The complete description is done in Subsection 8.

3.2 The SimDetect Method

Detecting the behavior of botnets and the behavior of bots are two quite different problems [8]. The behavior of one Bot can be detected by means of the analysis of one host [73]. The behavior of a Botnet can be detected, for example, based on the synchronization between the individual bots [74]. Our technique focuses on differentiating bot behavior among non-botnet traffic. Figure 3.1 shows a schema of the SimDetect method summarizing all the steps.

The first step of our method consists in capturing network data from infected computers and normal computers. This was done with the tcpdump tool [75], which is a standard tool for capturing packets. The capture was done very cleanly, not loosing packets and not capturing traffic that is unrelated with the infected machine. The resulting pcap files are considered the dataset for the SimDetect method.

The second step processes the dataset by separating the TCP flows and extracting some information about them. We used a tool called tcptrace [76] for this purpose. This tool generates a text file with all the characteristics of every TCP flow. The output files of the tcpreader tool were processed to extract the features we need of each flow. We use our own custom tool to extract the following features: start timestamp in UNIX epoch time, source IP address in decimal format, destination IP address in decimal format and destination port. This tool also adds to each flow the label *botnet* if it corresponds to an infected machine and *non-botnet* if it corresponds to a non-infected machine.

Since this method tries to detect the synchronization inside the bots traffic, the third step involved the aggregation of the flows in one-second time windows. Each time window contains aggregated information about every TCP flow within it. For every TCP flow in each time window, we compute the amount of unique source IP addresses, unique destination IP addresses and unique destination ports. This information is aggregated in an attempt to reduce the data processing complexity and also to save only the useful information. After computing the features each time window can then be represented by a four dimensional vector containing the window Id, amount of unique source IP addresses in that time window, amount of unique destination IP addresses in that time windows and amount of unique destination TCP ports in that time window. An example vector should look like this: [23, 1, 10, 1]. These vectors were stored in files with the ARFF format (Attribute-Relation File Format).

The fourth step involves clustering these vectors using the EM (Expectation-Maximization) algorithm [77]. The Expectation-maximization is a method for finding the maximum likelihood estimates of parameters in statistical models with incomplete data, and has been used successfully before in traffic flow analysis for characterizing communication connectivity patterns [78], botnet detection [79] and spam detection [80].

A basic introduction to the EM algorithm follows. We have a density function $p(x|\theta)$ that is defined by a set of parameters θ. We also have a known dataset $X = x_1, \ldots, x_n$, of size N, extracted from this distribution. The resulting density for the samples can be computed using Equation 3.1.

$$p(X|\theta) = \prod_{i=1}^{N} p(x_i|\theta) = L(\theta|X) \tag{3.1}$$

This function L is called the likelihood of the parameters given the known data. The likelihood is a function of the parameters θ where the known data X is fixed. In the maximum likelihood problem, represented by Equation 3.2, the goal is to find the θ parameter that maximizes L.

$$\theta^{'} = arg \max_{\theta} L(\theta|X) \tag{3.2}$$

The EM algorithm is a technique to solve the maximum likelihood problem. In our proposal, the windows vectors are the known data, coming from different statistical distributions (botnets). The hidden variables to be found are how many distributions there are and which vectors correspond to each distribution. Each botnet generates packets given a unique statistical distribution with unknown parameters θ and our goal is to maximize the a-posteriori probability of these parameters given the known data (vectors) in the presence of hidden data.

The intuition of EM is to alternate between estimating the unknowns θ and the hidden variables. The idea is to start with a guess of these parameters θ and to determine for each vector which underlying distribution is more likely to have generated the observed data (using the current parameters estimates). Then, assume these guessed assignments to be correct and apply the regular maximum likelihood estimation procedure to get the next θ. The procedure iterates until convergence to a local maximum. The algorithm computes the models parameters and assigns the cluster number with highest probability to each window vector [81].

The Expectation-Maximization algorithm provides a simple, easy to implement and efficient tool for learning the parameters of the model. The algorithm was implemented using the Weka framework [82] by reading the ARFF file with all the aggregated data. The EM implementation in Weka was selected because of its independence assumption of the attributes in the model. Weka adds a new column to the ARFF file with the cluster number. Once the clustering is finished, we computed the percentage of botnet and non-botnet instances in each cluster. We can then compute how many botnets and non-botnets instances were assigned to each cluster.

We define clustering as the task of grouping a set of objects in such a way that the objects in the same group (cluster) are more similar (depending on the definition of similar) to each other than to other objects in other clusters.

Two important considerations should be made about the SimDetect method. First, that it is designed to detect botnet clusters; not to differentiate between botnet and *normal* clusters. The difference is important because what is not considered to be a botnet cluster can be more than just *normal*. For example the clusters that are not botnet could be manual attacks, automatic attacks, port scans and also normal traffic. The next Subsection describes the experiments to evaluate the SimDetect method.

3.3 Experiments

This Section describes the experiments conducted to verify the SimDetect method. They were ran over a new dataset of botnet traffic that was specially gathered for this method. The dataset was created using three different testbeds that generated six different captures. Three of these captures contain botnet traffic and three captures contain non-botnet traffic. The testbeds and the captures are completely described in Subsection 8.2. The three malware captures are: *botnet1*, that is a *Virut* malware; *botnet2*, that is an *Agobot* malware; and *botnet3*, that is an *Rbot* malware. Also we obtained two normal captures from a university network and a non-botnet capture with a port scanning.

Each of the experiments to verify the SimDetect method was run in a combined dataset from one botnet capture and one non-botnet capture. In this way the experiments had a more real setup to work. The merge of the botnet and non-botnet captures was done by concatenating both tcptrace text files and modifying their start times. This approach proved easier than modifying and mixing pcap traces.

The selection criteria for both types of captures for each experiment had two steps. In the first step we selected the captures based on how much they represent its own class. Captures with more non-botnet applications were preferred and botnets with more traffic rates were preferred. In the second step we decided how to mix both types of captures based on the assumption that mixed traffic is more difficult to differentiate than concatenated traffic. When non-botnet and botnet traffic are only concatenated and not superimposed, the separation of both classes is straightforward. The mix of both types of traffic was done by inserting the smaller capture into the larger one. In this way we obtained a dataset with a good mixture of traffic.

The decision to mix the captures in the dataset is motivated by the fact that Non-botnet computers tend to have low traffic generation rates, whereas bots tend to have bursts of high traffic rates [83]. This difference can be as much as three times greater. If the captures are not correctly mixed, the resulting clusters may have an uneven number of

Cluster #	Botnet Flows	Non-Botnet Flows	Botnet %	Non-Botnet %
0	2,128	0	100%	0%
1	7,129	0	100%	0%
2	5,633	0	100%	0%
3	0	7,971	0%	100%
4	480	0	100%	0%
5	240	0	100%	0%
6	3,322	0	100%	0%
7	2,192	0	100%	0%
8	0	37,177	0%	100%

TABLE 3.1: Detection percentages of 45,148 Non-botnet1 flows mixed with 21,124 Botnet2 flows

packets. This may result in natural *unbalanced* experiments. If we have a balanced and mixed data set (the number of samples in botnet and non-botnet classes is almost equal), the classifier error rate may be more correct since each instance will belong to each class with 50% probability approximately. If we have an unbalanced data set (the number of samples in different classes varies greatly) the error rate of a classical statistics classifier could not represent the true effectiveness of the algorithm.

To verify the SimDetect method we ran four experiments. Each of them dealt with a different combination of datasets to represent several botnet situations. The first experiment was an attempt to differentiate botnet traffic from port scanning traffic. This is an important separation because port scanning activities are a common and valid network administration tool and at the same time are a common phase in any network attack. This experiment analyzed a capture that is the concatenation of the port scanning flow from the capture *Non-botnet1* and the bot traffic from the capture *Botnet2*. Table 3.1 shows the details of the results. It can be seen that the higher traffic rate of the port scans resulted in a very clear traffic separation. These results suggests that botnet traffic and network scans have different network behaviors.

The aim of the second experiment, shown in Table 3.2, was to analyze the Rbot malware within the *Non-botnet3* capture. This capture is composed of one non-botnet computer alone. Assuming that a computer behaves almost identically during fourteen days of office work, the original four-hour non-botnet capture was copied twenty times, making each copy start in a different day at the same hour. We finally had 19,443 non-botnet flows from September 13 to October 2 and 34,118 botnets flows from September 14 to September 15. After aggregating the flows into one-second time windows, we generated the ARFF file for this traffic and processed it with the EM algorithm. This process generated four clusters. This experiment tried to differentiate between 19 days of low rate non-botnet traffic and ten hours of a high rate bot burst in the middle of it.

Cluster #	Botnet Flows	Non-Botnet Flows	Botnet %	Non-Botnet %
0	14,433	24	99.83%	0.16%
1	1,517	18,721	7.49%	92.50%
2	3,539	14	99.60%	0.39%
3	14,629	684	95.53%	4.46%

TABLE 3.2: Detection percentages of 19,443 Non-botnet3 flows mixed with 34,118 Botnet3 flows

Cluster #	Botnet Flows	Non-Botnet Flows	Botnet %	Non-Botnet %
0	2,147	1,358	61.25%	38.74%
1	3,539	12	99.66%	0.33%
2	14,469	81	99.44%	0.55%
3	13,963	66	99.52%	0.47%

TABLE 3.3: Detection percentages of 1,517 Non-botnet2 flows mixed with 34,118 Botnet3 flows

Cluster #	Botnet Flows	Non-Botnet Flows	Botnet %	Non-Botnet %
0	2,147	916	70.09%	29.90%
1	13,918	22	99.84%	0.15%
2	3,539	14	99.60%	0.39%
3	14,469	25	99.82%	0.17%
4	45	0	100%	0%

TABLE 3.4: Detection percentages of 977 Non-botnet3 flows mixed with 34,118 Botnet3 flows

It also analyzed how the algorithm performed with an almost even number of packets in both classes.

In the third experiment, shown in Table 3.3, the goal was to analyze an unbalanced and more realistic situation. In this experiment we mixed the three hours of the *Non-botnet2* capture with ten hours of bots traffic from the *Botnet3* capture. The non-botnet capture was modified to start on September 15, in the middle of botnet traffic. This experiment was good to analyze an IRC bot with a normal computer.

In the fourth experiment, shown in Table 3.4, we analyzed one infected computer on the *Botnet3* capture among a network of non-botnet computers from the *Non-botnet3* capture. The five hours-long non-botnet capture was modified to start on September 15, in the middle of the ten-hour botnet traffic. Non-botnet traffic was composed of more than 40 office computers. This is a very realistic setup, where many computers generate very low traffic rates.

3.4 Error Metrics and Analysis of the Results

In any detection experiment it is important to report the results, but it is also very important to analyze these errors correctly. This Subsection analyzes the error rates of our experiments. The classic errors definitions are used for the flows, i.e. a false positives is counted each time a non-botnet flow is detected as botnet, and a false negative is accounted each time a botnet flow is detected as non-botnet.

A quick analysis of the effectiveness of our algorithm can be done by estimating the false positives and false negatives for each IP address. Table 3.5 shows some basic statistics for each IP address that was erroneously detected. The Table includes the number of times the flows from each IP address were detected as botnet and as non-botnet. The IP addresses in this table are representative of the most active computers in the experiments. The first experiment was not included in this Table because its goal was to prove that the botnet traffic and the intrusion attempts could be successfully separated.

In Table 3.5 it can be seen that the FPR of the second experiment is reasonably low. In the second experiment there is only one false negative, where one IP address is detected as non-botnet 6.45% of the time. A detailed manual analysis showed that the non-botnet label was assigned to this botnet when a single flow per second was generated. This is an example of a botnet behaving like a non-botnet computer. As IP address 10.1.1.1 was the network web proxy and therefore it may be white-listed by the administrator, it may be possible to achieve a better error rate.

The third experiment has, comparatively, higher false positive error rates. An analysis of these errors shows that both botnet and non-botnet computers were clustered together because they generated several connections at the same time. This situation was a trigger to further analyze the botnet behaviors and it was addressed in our other botnet detection methods.

The fourth experiment shows an increment in false negatives values and its false positive error rate is notably higher because fewer instances were used. In the worse case, 6.18% of the time a non-botnet computer is detected as botnet. This experiment did no show any False Negative error.

The information on Table 3.5 may indicate that the most challenging problem may be how to correctly detect a non-botnet computer traffic that behaves like a botnet. A confusion matrix was created from the data in this Table to better analyze the meaning of the error rates. It is shown in Table 3.6. A confusion matrix is a visualization tool commonly used in supervised learning, where each row represents how each instance was detected and each column what it actually was.

Exp.	Error Type	IP Address	Actually is	# Detected as Non-Botnet	# Detected as Botnet	Error %
2	FP	10.1.1.1	Non-Botnet	10,570	180	1.67%
2	FP	192.168.2.79	Non-Botnet	18,697	542	2.81%
3	FP	10.1.1.1	Non-Botnet	466	1	0.21%
3	FP	192.168.2.71	Non-Botnet	5	1	16.66%
3	FP	192.168.2.74	Non-Botnet	783	89	10.20%
3	FP	192.168.2.76	Non-Botnet	235	31	11.65%
4	FP	192.168.2.79	Non-Botnet	910	60	6.18%
2	FN	192.168.3.104	Botnet	1,103	15,978	6.45%
3	FN	192.168.3.104	Botnet	1,563	15,518	9.15%
4	FN	192.168.3.104	Botnet	1,563	15,518	9.15%

TABLE 3.5: IP addresses error evaluation

In total, the SimDetect method processed 124,291 different instances. The False Positive Rate (FPR) is the proportion of non-botnet flows that yield a botnet outcome (904) among all the tests performed ($904 + 21,033$). The value of FPR achieved was 0.04. The False Negative Rate (FNR) is the proportion of botnet flows that yield a non-botnet outcome (4,229) among all the actual botnets ($98,125 + 4,229$). In our case the FNR was 0.04.

Some technical results can also be extracted from Table 3.6, for example, 0.91% is the percentage of non-botnet flows that were detected as botnet among the presumed botnets, i.e., from the total amount of infection alerts that the network administrator receives from our algorithm, only 0.91% were not infected.

From Table 3.6 we can also conclude that 16.74% is the proportion of botnet flows that were detected as non-botnet among the presumed non-botnet computers. This would be the proportion of infected flows that the administrator will not see. Note that we are talking about flows and not computers. An analysis from the computer point of view is done later.

Still from a technical perspective, if this algorithm was used in real botnet shutdown campaigns, 0.04 of the bots will not be detected and will remain active. Finally, we are going to bother 0.04 of our internal users trying to clean their non-infected computers.

	Actually Botnet	Actually Not Botnet	
Detected Botnet	TP=98,125	FP=904	0.91%
Detected Not Botnet	FN=4,229	TN=21,033	16.74%
	FNR=0.04	FPR=0.04	

TABLE 3.6: Confusion Matrix for the detection of botnet Flows

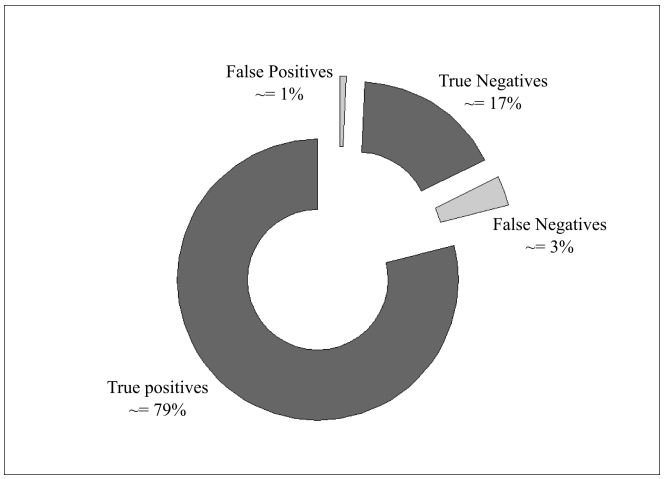

FIGURE 3.2: Error Analysis

Figure 3.2 is a graphical analysis of the confusion matrix. It shows the approximate percentages of the errors and also the relation between errors and true detections.

3.5 Conclusions of the SimDetect Method

In this chapter we presented a new approach for bot network behavior detection using the Expectation-Maximization clustering algorithm. Our work is based on the hypothesis that bots behave different than non-botnets computers. To analyze this situation, we captured several network flows with botnet and non-botnet traffic using three different test beds. This traffic included single and network captures of both current malware and not infected computers. After proper traffic pre-processing, we separated the traffic flows in time windows and studied the relationship between unique source IP addresses, unique destination IP addresses and unique destination ports within these time windows. The Error rates and results were encouraging, showing that bots can be differentiated from non-botnet traffic with a FPR of 0.7% and a FNR of 3.4%.

The SimDetect method presents some advantages from other proposals. First, it can detect bots even if the traffic is encrypted. Second, it does not need the payload of the protocol to accomplish the detection or generate signatures. Third, the separation in time windows allows it to detect bots within the first stages of the infection. Our second hypothesis is that a bot could be detected because of the high amount of connections generated in a time window. The preliminary experimental results suggest that this assumption is useful under certain conditions, because botnet and non-botnet traffic showed different network characteristics that may helped to separate them in clusters. The disadvantages of this method are, first, that the features used were not good enough

to separate the botnet traffic in all the experiments. Second, the method can not detect the botnet clusters automatically.

This chapter makes the following contributions to our general analysis of botnet behavior.

- The separation of aggregated network flows in time windows seems to be a promising technique for analyzing the behavior of botnets.
- The Expectation-Maximization clustering method gave good results that may be extended.
- The labeled dataset was very important in the analysis of the errors and the improvement of the method.

Chapter 4

The BClus Method: Clustering Botnet Behavior

4.1 Introduction

The previous chapter described our first attempt to detect botnets and bots with the creation of the SimDetect method. Based on the good results obtained we learned that the use of a clustering method seemed to be a good option for getting all the similar traffic together. However, its main drawback was that the detection was not automatic. Based on the clustering idea and to solve the automation issue we designed a new detection method called BClus. Regardless that both methods use clustering algorithms, the BClus method is very different and far more complex than SimDetect.

The design of the BClus method is based in the idea that the aggregated information in the network should be separated by source IP address. Once clustered, this information feeds a decision tree classification algorithm, which is responsible for automatically detecting the botnets. The BClus method can detect the behavior of botnets the network based on how known botnet clusters look like.

During the creation of the BClus method we became conscious of the importance of having a good and labeled dataset. It is known that without the proper data and, more importantly, without the proper labels, it is very difficult to create a good detection method [9]. For the creation and evaluation of the BClus method we created a very large and real botnet dataset. The main characteristics of this dataset are that it is completely labeled with Background, Normal and Botnet labels, that it is real, large and publicly available. This may be the most complete botnet dataset available to date. Its

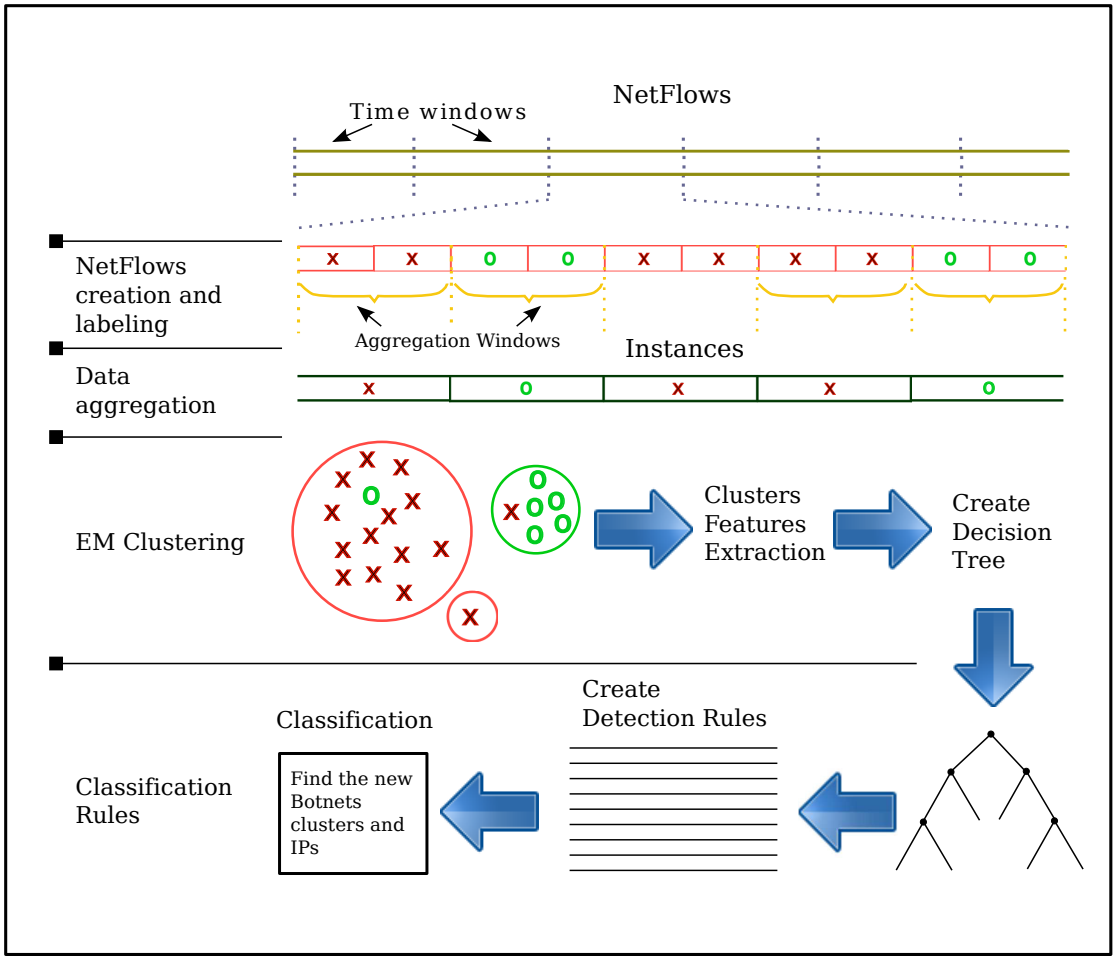

FIGURE 4.1: Summary schema of the BClus detection method.

creation was possible due to a join project in cooperation with the CTU University of Czech Republic.

For the purpose of the BClus method this dataset was processed and separated in training and testing scenarios. The complete description about the dataset is done in the Subsection 8.3. The following subsections describe the inner working of the BClus detection method.

4.2 The BClus Detection Method

The BClus method is a behavioral-based botnet detection approach. It creates models of known botnet behavior and uses them to detect similar traffic on the network. It is not an anomaly detection method. Figure 4.1 shows a summarized schema of the BClus method.

The purpose of the method is to cluster the traffic sent by each IP address and to recognize which clusters have a behavior similar to the botnet traffic. A steps of the BClus method are:

1. Separate the NetFlows in time windows.
2. Aggregate the NetFlows by source IP address.
3. Cluster the aggregated NetFlows.
4. Only during training: Assign ground-truth labels to the botnet clusters.
5. Only during training: Train a classification model on the botnet clusters.
6. Only during testing: Use the classification model to recognize the botnet clusters.

At the end the BClus method outputs a predicted label for each NetFlow analyzed. The following Subsections describe each of these steps.

4.2.1 Separate the NetFlows in Time Windows

The main reason to separate the NetFlows in time windows is the huge amount of data that has to be processed. Some of the datasets used in this approach produced up to 395,000 packets per minute. A short time window allows us to better process this information.

The second reason to use time windows is that botnets tend to have a temporal locality behavior [84], meaning that most actions remain unchanged for several minutes. In our dataset the locality behavior ranges from 1 to 30 minutes. This temporal locality helps capture all the important behaviors in the time window.

The third reason to use time windows is the need to deliver a result to the network administrator in a timely manner. After each time window, the BClus method outputs some results and the administrator can have an insight about the traffic.

An important decision on the time window separation criteria is the window width. A short time window does not contain enough NetFlows and therefore would not allow a correct analysis of the botnet behavior. In the other hand, a large time window would impose a high computational cost. The time window used by the BClus method is of two minutes, since it is enough to capture all the botnet behaviors and it does not contain too many NetFlows.

The next step in the BClus method is to aggregate the NetFlows.

4.2.2 Aggregate the NetFlows by Source IP Address.

The purpose of aggregating the NetFlows is to analyze the problem from a new high-level perspective. The aggregated data may show new patterns. We hypothesize that these new patterns could help recognize the behaviors of botnets. From the botnet detection perspective, the main motivations for aggregating NetFlows are the following:

- Each bot communicates with the C&C server periodically [85].
- Several bots may communicate at the same time with the same C&C servers [45].
- Several bots attack at the same time the same target [86].

Inside each time window, the NetFlows are aggregated during one *aggregation window*. The width of the aggregation window should be less than the width of the time window, which is of two minutes. After some experimentation, a one minute aggregation window width was selected, which is is enough to capture the botnet synchronization patterns and short enough not to capture too much traffic [29]. Therefore, on each time window, two aggregation windows are used.

The NetFlows are aggregated by unique source IP address. The resulting features on each aggregation window are:

1. Source IP address
2. Number of unique source ports used by this source IP address.
3. Number of unique destination IP addresses contacted by this source IP address.
4. Number of unique destination ports contacted by this source IP address.
5. Number of NetFlows used by this source IP address.
6. Number of bytes transferred by this source IP address.
7. Number of packets transferred by this source IP address.

We call this group of seven aggregated features an *instance* to simplify the references. The aggregation step ends with a list of instances for each aggregation window. The next Subsection describes the clustering process of these instances.

4.2.3 Cluster the Aggregated NetFlows

The continuous evolution of botnets suggests that a good detection method should be as independent of the network characteristics as possible. The BClus method, then, uses an unsupervised approach to cluster the instances described in the previous section. These natural groups of behaviors depend on the time window being analyzed and on the characteristics of the network where the algorithm is executed.

The technique used for this task is WEKA's implementation of the Expectation-Maximization (EM) algorithm [87]. Our dataset has many different network behaviors generated by normal, botnet and attack actions. We hypothesize that these behaviors are generated from different probabilistic models and that the parameters of these models can be found using the EM algorithm. The instances are assigned to the probability distribution that they most likely belong to, therefore building clusters.

After generating the clusters, the task of the BClus method is to find which of them belong to botnets. To find out which of the generated clusters belong to botnet actions, we compute some features of each cluster and then train a classification algorithm to learn how to recognize them. The features stored for each cluster are the average and standard deviation of the features of the seven instances described in Subsection 4.2.2. The final 15 cluster features are:

1. Total number of instances in the cluster.
2. Total number of NetFlows in the cluster.
3. Number of source IP addresses.
4. Average number of unique source ports.
5. Standard Deviation of the number of unique source ports.
6. Average number of unique destination IP addresses.
7. Standard Deviation of the number of unique destination IP addresses.
8. Average number of unique destination ports.
9. Standard Deviation of the number of unique destination ports.
10. Average number of NetFlows.
11. Standard Deviation of the number of NetFlows.
12. Average number of bytes transferred.

13. Standard Deviation of the number of bytes transferred.
14. Average number of packets transferred.
15. Standard Deviation of the number of packets transferred.

Once the features are extracted for each cluster, they are used in the next Subsection to assign the ground-truth labels to the clusters.

4.2.4 Train a Classification Model on the Botnet Clusters

Once we extracted all the features for the clusters, we need to learn how to recognize the botnet clusters. The idea is that the values of the features and the relationship between the features for the botnet clusters are different from the normal clusters. The complete steps in our training phase are:

1. Use a leave-one-out algorithm with the training and cross-validation datasets. For each round do:
   (a) Separate the NetFlows in time windows. (Subsection 4.2.1)
   (b) Aggregate the NetFlows by source IP address. (Subsection 4.2.2)
   (c) Cluster the aggregated NetFlows. (Subsection 4.2.3).
   (d) Assign ground-truth labels to the clusters based on the ground-truth labels of the NetFlows. (Subsection 4.2.4.1).
   (e) Train a JRIP classification model to recognize the botnet clusters.
   (f) Apply the JRIP model in the cross-validation dataset of this round.
   (g) Store the error metrics of this round.
2. Select the best JRIP model based on the results of the leave-one-out.
3. Testing phase. (Subsection 4.2.5).
   (a) Read the testing dataset.
   (b) Use the best JRIP classification model to recognize the botnet clusters.
   (c) Assign the labels to the NetFlows based on the labels of the clusters.
   (d) Calculate the final error metrics.

The classification algorithm used to find the botnet clusters is JRIP. This algorithm is the WEKA's implementation of a "(...) a propositional rule learner, Repeated Incremental

Pruning to Produce Error Reduction (RIPPER), which was proposed by William W. Cohen as an optimized version of IREP" [82]. The JRIP algorithm receives a labeled group of clusters and outputs a group of rules to detect them. The definition of classification that we consider is the problem of identifying to which category a new observation belongs, based on a training dataset with known categories. The rest of the Subsections describe each of the remaining steps.

4.2.4.1 Assign Ground-Truth Labels to the Botnet Clusters

The ground-truth labels should be assigned to the clusters because we need to train and evaluate the JRIP classification algorithm with them. Once the JRIP algorithm knows which the botnet clusters are, it can create a model to recognize them.

To assign a ground-truth label to a cluster, we should first assign a ground-truth label to all of its instances (aggregated NetFlows). However, to assign a ground-truth label to an instance, we should first assign a ground-truth label to all of its NetFlows.

Since the dataset used for this detection method was labeled by hand by experts, we have the labels of each NetFlow. Therefore, the ground-truth label of each instance is known, because an instance is composed of all the NetFlows from the same IP address. However, a cluster is composed of *different instances* coming from *different source IP addresses*. Then, it is not straightforward to know which ground-truth label should be assigned to a specific cluster. This Subsection describes the process that we used to assign a ground-truth label to each cluster.

To help us decide which label should be assigned to each cluster, a new *informational* feature was computed for each cluster: The percentage of botnet NetFlows on the cluster. This value is expected to be bigger on botnet clusters and smaller on background clusters and it was used to select the ground-truth label for the cluster. Notice that this feature is only used to assign the ground-truth label in the training phase and is neither stored nor used in the testing or training phase.

As this new feature is a percentage, a correct threshold had to be found. This threshold decision is very important, because different percentages correspond to different botnet behaviors. If it is above 0%, it means that every cluster with at least one botnet NetFlow is considered a representative of a botnet behavior. If it is above 1%, it means that only clusters with more that 1% of botnet NetFlows are considered as representative of a botnet behavior. A manual analysis of the dataset determined that most of the real botnet clusters had between 0% and 1% of botnet NetFlows. To find out which threshold between 0% and 1% was the best, we implemented the leave-one-out algorithm

described in Subsection 4.2.4 to try the following ten candidates thresholds: 0.1%, 0.2%, 0.3%, 0.4%, 0.5%, 0.6%, 0.7%, 0.8%, 0.9% and 1%.

After running the leave-one-out we found that the group of clusters that has the best error metrics for the BClus algorithm was generated with a threshold of 0.4%. The set of JRIP rules generated by the 0.4% percentage became the best detection model that is applied in the next Subsection.

4.2.5 Testing phase. Use the Classification Model to Recognize the Botnet Clusters

Once the best detection model was found in the previous Subsection, we applied it on each of the testing datasets to obtain the final results. The testing methodology is the following:

1. Separate the NetFlows in time windows.
2. On each time window, aggregate the NetFlows in aggregation windows (now called instances).
3. Cluster the instances with the EM algorithm.
4. Extract the 15 features of each cluster.
5. Use the JRIP decision rule algorithm learned in the previous Subsection to decide if a cluster is a botnet cluster or not.
6. If a cluster is detected as a botnet cluster, then label all of its instances as botnet and label all of the NetFlows as botnet.
7. Output the original NetFlow file with the new predicted labels.

With the results for each testing dataset obtained, we can now compute the error metrics of the BClus Method.

4.3 Results and Analysis

The BCLus detection method was executed on each of the five testing datasets described in Section 8.3. Each execution of the method generated a NetFlow file with the predicted labels. Using these predicted labels and the original ground-truth data we computed

Scen.	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
1	30.2	41.3	27.6	35.3	0.4	0.5	0.4	0.5	0.5	0.5	0.4	0.48
2	15.6	37.1	9.8	34.2	0.3	0.7	0.2	0.6	0.6	0.5	0.4	0.41
6	0.6	20.2	0.7	28.6	0.0	0.9	0.0	0.9	0.4	0.4	0.5	0.04
8	23.5	152.8	77.	209.9	0.1	0.6	0.3	0.8	0.2	0.3	0.6	0.14
9	10.1	46.4	11.5	48.2	0.1	0.8	0.2	0.8	0.4	0.4	0.5	0.2

TABLE 4.1: Results of the BClus method on each of the five testing scenarios. The tTP, tTN, tFP and tFN are absolute values. The TPR, TNR, FPR, FNR, Prec, Acc, ErrR and FM1 values are percentages.

several error metrics. The error metrics can be used to have an idea of the performance of the algorithm and to analyze the output of the method. However, a deeper analysis and comparison of these metrics with other detection methods is done in Chapter 7.

The error metrics are presented in the Table 4.1. Scenario 1 corresponds to an IRC-based botnet that sent spam for almost six and a half hours. The Scenario 2 executed the same IRC-based botnet as scenario 1, but in this one the malware sent SPAM for 4.21 hours. The botnet in Scenario 6 scanned SMTP (Simple Mail Transfer Protocol) servers for two hours and connected to several RDP (Remote Desktop Protocol) services. However, it did not send any SPAM and did not attack. The C&C server used a proprietary protocol that connected every 33 seconds and sent an average of 5,500 bytes on each connection. In Scenario 8 the botnet contacted a lot of different C&C hosts and received large amounts of encrypted data. It also scanned and cracked the passwords of machines using the DCERPC protocol both on the Internet and on the local network for 19 hours. In Scenario 9 ten hosts were infected using the same Neris botnet as in Scenarios 1 and 2. For five hours, more than 600 SPAM mails were successfully sent.

The results presented in the previous Section are promising and challenging. At first glance it may seem that the results on Table 4.1 have small TPR values or that the FPR values are high. However, these datasets are composed of millions of flows with a specific balance of Background, Normal and Botnet labels. This means that the datasets are very hard to analyze and that it may be really easy to make mistakes, since they have real labeled data. To have a better idea of how good the results are we need to compare them with other detection methods. This comparison is done in the Chapter 7.

4.4 Conclusions of the BClus Method

The BClus method presents a great advance in the detection of botnets in the network compared to our previous detection method. It can automatically detect botnets based on the clustering of similar behaviors of botnets and based on a decision tree algorithm.

The core of the method was trained to recognize how does a cluster of botnet-generated NetFlows looks like based on a predefined group of features. The results are very encouraging. However, to be complete these results need to be compared with other detection methods using the same dataset. That is the goal of the next Chapter.

The key highlights of the BClus method are the usage of a big and labeled dataset, the application of a classification algorithm on the features of clusters and the automatic detection on botnets.

Chapter 5

A Deep Behavioral Analysis of the Botnets CC Channels

5.1 Introduction

The development and comparison of the BClus detection method gave us several insights on how to improve the detection techniques. The error analysis and the detailed study of the botnet actions showed us that there are identifiable behaviors in the traffic of most botnet connections, and in particular in the traffic of the C&C (command and control) channels. A C&C channel is any mean of communication that the bot uses to send data back to the botmaster and to receive orders. Usually the C&C channels are implemented using the HTTP, UDP or TCP protocols. The understanding of these botnets behaviors may provide important clues on how to improve our detection method. The goal of this Chapter is to analyze a large botnet capture to characterize, understand and model the behavior of its C&C channels.

The behavioral analysis of any botnet traffic has to deal with the actions, connections and patterns of botnets over time. However, these patterns can be so complex and interdependent that they may only by seen by analyzing a long-term capture [88]. Most botnets are use several different modules for their actions, and each module produces a different behavioral pattern on the network. By analyzing each pattern and correlating the actions it may be possible to understand how botnets work. To capture all these behaviors we need a large capture, so the patterns have enough time to emerge.

For the analysis we use a malware dataset that was captured in the Malware Capture Facility Project (MCFP) [89] in the CTU University. This malware corresponds to a possible variant of the Zbot family and it lasted 57-days. During the analysis we

extracted the characteristics of all its C&C channels and we created states models of each network behavior.

The contributions of this Chapter are:

- A deep analysis of the behavioral characteristics that distinguish a botnets C&C channels. (Section 5.2).
- A state-based model of each C&C channel and of the botnet as a whole entity. (Section 5.2).
- An analysis of the relationship between the channels and the botnet actions.

5.2 Behavioral Analysis of the C&C channels

The variant of the Zbot (Zeus) Botnet analyzed in this chapter performed several actions such as downloading binary files and connecting to Google among others. It also communicated with its Botmaster using different protocols. Each of the actions generated a different behavior on the network traffic during the 57 days of operation. From these behaviors, the C&C channels are considered to be the most important [85], because they typically show very specific features that may be used to detect the botnet. The most relevant are that the C&C channels last for several days continuously, they are present long before the attacks are started, and they must be used by the Botmaster in order for the bot to be useful.

The C&C channels on the Zbot capture need to be manually detected and labeled before they are used to build a behavioral model. However, it is not easy to find out which traffic corresponds to a C&C and which it does not. The ad-hoc procedure that we used to identify them is based on the following assumptions: first, the traffic of every bot is part of some malicious actions or is part of its communications channels. Second, most C&C channels have an almost periodic behavior [85]. Third, the C&C channels are usually encrypted [16] since if they are not, their purpose can be easily identified. Using these assumptions we were able to manually find out the C&C channels in the traffic by analyzing the network packets.

For the analysis of the data in network, e.g. the DNS traffic, it is very useful to study the packets. However, to analyze other features, like the periodicity, it is easier to aggregate the packets in network flows. In previous Chapters we work with and explain the notion of NetFlow. From this Chapter on we will stop using NetFlows and we will use a broader definition of flow, which is similar but not the same. This change was

done because Cisco's definition of NetFlow does not include important features that are needed for our research. Instead, we use the much richer definition of flow provided by the Argus suite [90]. The differences between the NetFlow standard (specially NetFlow v5) and the Argus flows are that the NetFlows are restricted to the version 4 of the IP protocol, and that they are unidirectional. For Argus, a flow is defined as all the packets that share the source IP address, the source port, the destination IP address, the destination port and the protocol. Apart from these features, three more definitions are needed to correctly separate the flow: the timeout of the flow, the report time of the flow and the directionality of the flow. The timeout is the inter-packet time that must pass to not consider the next packet as part of the same flow anymore. The report time is how often the traffic processing application sends the information about the flows to the flow receiver. The directionality of a flow is an indicator of weather one flow contains all the packets for both IP addresses or if it contains the traffic on only one IP. These three timing definitions are needed to correctly define what a flow is. As we used the Argus toolkit for generating the flows, we also used its default timeout values and its bidirectional flows.

Using our manual analysis of packets and flows, several different C&C channels were found during the analysis of the dataset, and three different states models were created for them. The following Subsections describe all of the C&C channels found: the UDP C&C channel, the TCP C&C channel and the HTTP C&C channel.

5.2.1 The UDP C&C Channel

The first C&C channel that we analyzed is only composed of UDP flows.These flows had a duration up to several days, but they were sent in groups. This means that instead of being uniformly scattered, they were only sent in groups with a separation of thirty minutes. These groups had between 1 and 175 flows each and most of the groups lasted one or two minutes. Thirty minutes after one group ended another one started. If a flow lasted more than thirty minutes, it appeared in several groups. Based on this information, two analyses were done. First, a time-based analysis of the changes in the groups of flows, and second, an analysis of each flow from its start to its end through multiple groups.

One of the important characteristics of this UDP C&C channel was that the original executable botnet binary already included a list of IP addresses to start the channel. They could be found because the bot contacted these IP addresses without any prior DNS requests.

Another important characteristic of the UDP C&C channel was that its traffic was not plain text. Therefore, we performed an analysis to evaluate if its was encrypted. The verification method that seemed more appropriate was the one implemented by Lyda et al. [91], that uses entropy to evaluate the type of content in malware binaries. The authors worked with malware executable files instead of traffic, but since our traffic can be stored in pcap files, we could extend their work. The pcap format is a binary file format to store the complete information of each network packet. The program that was sued by Lyda et al. and that we used for the verification was ent[1], which computes the entropy, chi-square test, arithmetic mean, the monte carlo value for Pi and the serial correlation coefficient. We compute those values on four set of packets. First, the packets on the UDP C&C channel, which we believe are encrypted. Second, the non-encrypted HTTP packets for the Google requests described in Section 5.3. Third, the DNS packets that we know are not encrypted. Fourth, the TCP Actions traffic packets that we think are encrypted and are described in Section 5.2.2. The idea is to compare the entropy reported by the ent program on these four set of packets with the entropy reported for verified encrypted files in [91]. After the computation, the DNS packets had an entropy of 5.299455 bits per byte, the Google HTTP traffic had an entropy of 6.517970 bits per byte, the UDP C&C traffic had an entropy of 7.705928 bits per byte and the TCP actions had an entropy of 7.961638 bits per byte. According to Lyda et al. the entropy of the DNS packets and the HTTP Google traffic is the same as the not encrypted text, which is true. Also according to Lyda et al, the entropy of the UDP C&C traffic and TCP Actions corresponds exactly with the entropy of encrypted traffic.

One of the important conclusions of finding if the traffic is encrypted is that we can now assume that the UDP C&C channel may be P2P traffic. This is because it consisted of small UDP flows that send probably encrypted data using the same source port. The next Subsection 5.2.1.1 describes the behavior of the UDP flows working as a group, and Subsection 5.2.1.2 describes the behavior of each UDP flow individually.

5.2.1.1 Behavior of the Groups of UDP Flows

The UDP flows in this capture can be analyzed from the point of view of their group behavior. The botnet sent a total of 2,674 groups of UDP flows. To analyze the patterns and relationships on these groups we separated and aggregated the traffic every thirty minutes. For each of these thirty minutes groups of flows, four features were extracted: first, the number of UDP flows that received a response and therefore were considered *established*. Second, the number of UDP flows that did not receive a response and therefore were considered *attempts*. Third, the number of *established* flows that had

[1] https://www.fourmilab.ch/random/

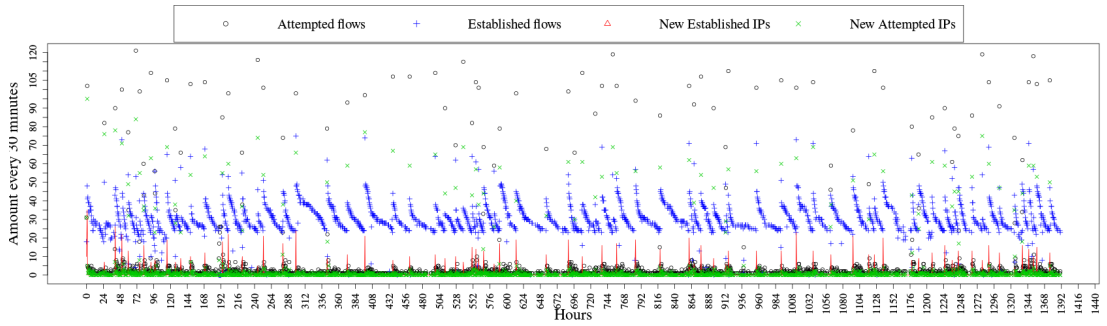

FIGURE 5.1: The number of UDP *attempted* flows, UDP *established* flows, new UDP *established* IP addresses and new UDP *attempted* IP addresses in the UDP C&C channel. Aggregation every 30 minutes.

new IP addresses compared to previous groups. And fourth, the number of *attempted* flows that had new IP addresses compared to previous groups. Figure 5.1 describe the relationship between these values. It shows the number of flows every thirty minutes for each of the four features.

The analysis of Figure 5.1 shows the patterns on the UDP C&C channel. The most visible behavior is the number of *established* flows, which starts around fifty flows and keeps decreasing until twenty flows approximately. Another behavior is that the number of *established* and *attempted* flows on each group differs. The groups mostly have a median of 29 *established* flows and 1 *attempted* flow, probably because the botnet had stable computers on the P2P network. However, some special groups had more *attempted* flows than *established*. These special groups are very important to understand the update mechanism of the C&C channel.

The special groups can be seen in Figure 5.1 as a lonely black circle in the upper part of the Figure. On those moments the botnet is sending the bot a new group of IP addresses. The bot seems to try each of these IP addresses and to add the ones that are answering. After these special groups which purpose seems to be to update the IP addresses, the bot has more *established* flows. An important observation is that, after some time, the number of *established* flows decreases. This can be seen in the Figure as a group of points that make a decreasing curved shape. This happens because not all the new IP addresses sent by the botnet are active and because some of the active ones become unreachable after some days. A probable explanation is that the peers in the network are also infected machines that are powered off or are being cleaned after some time. Two things can be learned from this behavior. First that the set of IP addresses is updated when the number of *established* flows is approximately under 23 and second, that the median time for this refreshment of IP addresses is 36 hours.

These moments when the botnet refreshes the list of IP addresses in the C&C channel are characterized by a peak of the four previous features: the number of *established*

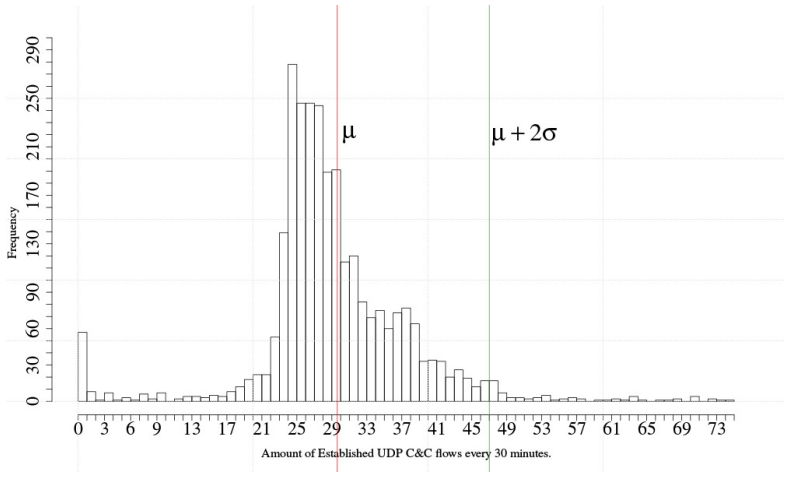

FIGURE 5.2: Histogram of the number of *established* flows every 30 minutes in the UDP C&C channel. Above the $\mu + 2\sigma$ line are the top flows.

flows with new IP addresses becomes greater than 0 (with a median peak of 10), the number of *attempted* flows with new IP addresses becomes greater than 1 (median of 23), the number of *attempted* flows becomes larger that 15 (median of 77.06) and the number of *established* flows becomes larger than 47 (median of 53.50). This threshold of 47 was needed to automatically find out which were the special groups and it was empirically selected by analyzing the histogram shown in Figure 5.2. This histogram shows the number of *established* UDP flows every 30 minutes. It can be seen that 47 is the first of the values that are more than two standard deviations above the mean ($\mu + 2\sigma$). On a normal distribution it is usually the top 2.4%. Although this is not a normal distribution, the threshold value was good enough to identify those moments. The groups of 30 minutes that have more that 47 flows, are considered top groups and represent the more active groups in the UDP C&C channel. Later on, these top groups will be compared to the other channels.

Another characteristic of the UDP C&C channel is how long lasted each *established* flow. For example, the first group of flows contained 18 IP addresses that were hard coded in the binary file. From these 18 IP addresses, only 4 had *established* flows. These 4 IP addresses remained *established* for a minimum of 12 hours and a maximum of 35 days. In summary, from the total of 5,699 unique IP addresses contacted inside the UDP C&C channel, only 1,101 were part of *established* flows (19.31%). From these 1,101 IPs, 75 (6.81%) were active between 10 and 20 days, and 31 (2.81%) were active more than 30 days. Only 11 (0.99%) were active more than 50 days. The only flow that remained *established* during the 57 days was to IP address 85.100.41.9 and destination port 8835. Figure 5.3 shows a histogram of the number of days that each flow lasted *established*.

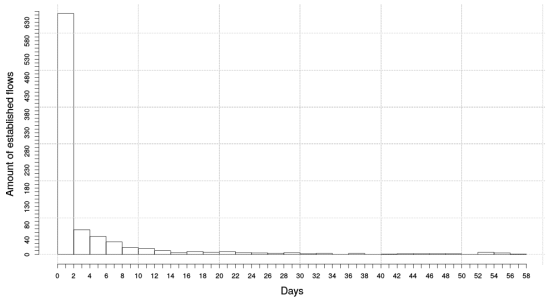

FIGURE 5.3: Histogram of the number of flows that remained *established* at least X days in the UDP C&C channel.

Type	tuples	Duration (sec)			Size (Bytes)			Time Diff (min)		
		Mean	Stdev	Med	Mean	Stdev	Med	Mean	Stdev	Med
Established	1,132	0.282	0.229	**0.226**	528.4	296	**431.4**	40.3	45.16	**30.6**
Attempt	5,906	0.018	0.031	0.006	204.1	**57.18**	200.5	411.7	384.8	310.9

TABLE 5.1: Differences in the duration (seconds), size (bytes) and periodicity (minutes) between the *established* and *attempted* 4-tuples in the UDP C&C channel.

5.2.1.2 Behavior of Each UDP Flow Individually

Besides the previously analyzed group behavior, each IP address also had its own behavior. To find them, the flows were grouped by the source IP, destination IP, destination port and protocol. This type of grouping is called *4-tuple*. When adding flows to a 4-tuple, we ignore the source port so we can focus on the *service* accessed by the bot. In this way, if the bot creates a new connection to the same service using another source port, the 4-tuple is still able to group the flows. A 4-tuple can be think of as a group of flows. It is a data structure that holds all the flows that share these 4 values.

To analyze its individual behavior, for each 4-tuple we computed the following summarizing features:

- Mean of the size of the flows.
- Stdev of the size of the flows.
- Mean of the duration of the flows.
- Stdev of the duration of the flows.
- Mean of the time differences between flows.
- Stdev of the time differences between flows.

These values can help identify a 4-tuple as part of the UDP C&C channel. A deeper description of the 4-tuples and how they were created is done in Subsection 6.2.2. Our analysis of the values of the features for the *established* and *attempted* 4-tuples showed that they were different. Table 5.1 shows that the *established* 4-tuples have a median time difference of 30.6 minutes that is stable (stdev of 45.16). On the other hand, the *attempted* 4-tuples have a median time difference of 310.9 minutes. This means that apart from the groups analysed on the previous Subsection, each *established* 4-tuple had a new flow every thirty minutes. The table shows that the median duration of the *established* 4-tuples was 37.6 times larger that the *attempted* 4-tuples, because the *attempted* flows did not get an answer. The table also shows that the median size of the *established* 4-tuples doubles that of the *attempted* 4-tuples, because of the number of responses. Finally, the stdev of the size of the *attempted* flows is low, showing that despite having more than five times the number of 4-tuples, its size was almost always the same.

Continuing with our analysis, we saw that some *established* flows sometimes had a different behavior. The first difference found was that the stdev of the time differences was low but not zero. This shows a variance of the time differences between 4 and 200 seconds (3.3 minutes) that may be characteristic of the C&C. The second difference found was that the time difference of the *established* 4-tuples was sometimes broken by a flow with a five seconds time difference. The third difference was that, from time to time, some *established* 4-tuples had larger flows. These larger flows were related to a change in the state of the C&C from idle to active. Larger flows mean more information going through the channel. For example, one 4-tuple sent 117 flows during 3 days with a size of 350 bytes. Then, it sent 32 flows with a size of 2,500 bytes during 16 hours. The difference can be seen as a small distribution of values around 2,300 bytes in the histogram of the sizes of the *established* flows in Figure 5.4.

5.2.1.3 The UDP C&C Channel States Model

The behavior of the UDP C&C channel shown in Figure 5.1 can be modeled as states. The goal was to identify a simple set of states that can represent the transitions found in the traffic. Figure 5.5 shows the state model created. The boxes represent states and the circles actions. The important states are the process to get new IP address in the channel and the change to an active C&C. The threshold to go from the Idle state to the Active state is the previously defined value of 47 *established* flows. This model is important to understand the basic behaviors of the channel and it was used to create a better detection algorithm.

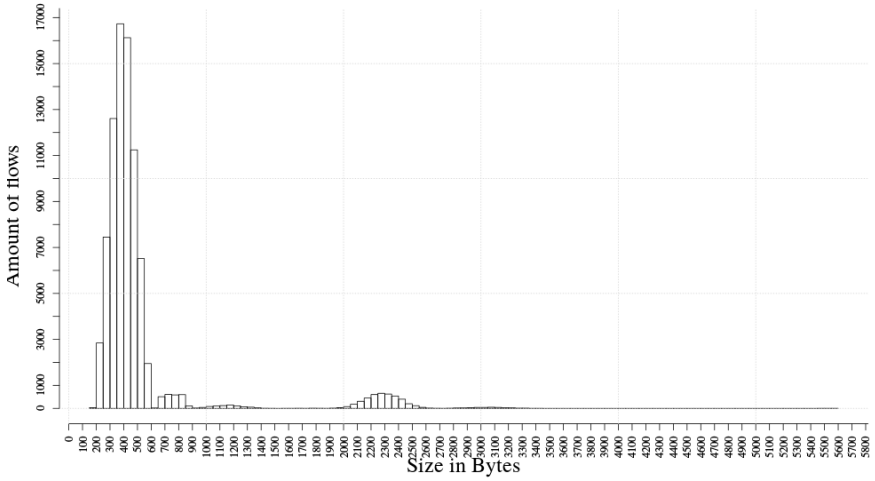

FIGURE 5.4: Histogram of the size in bytes of the *established* flows in the UDP C&C channel.

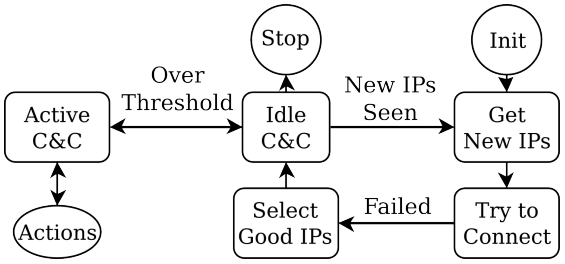

FIGURE 5.5: State Model for the UDP C&C channel.

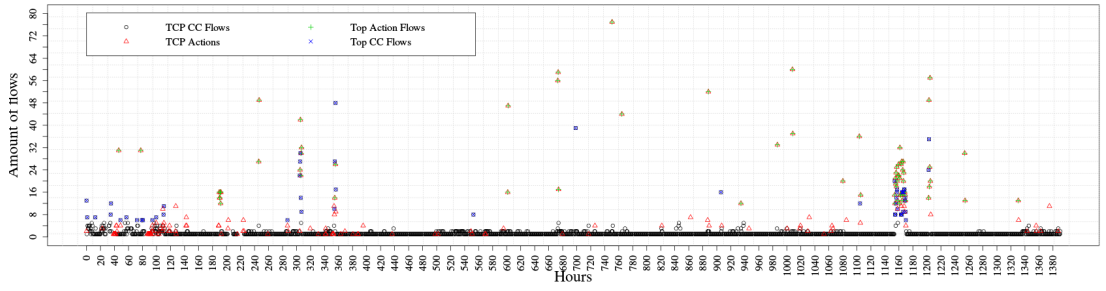

FIGURE 5.6: Comparison of the number of flows between the TCP C&C channel and the TCP action flows (Flows that look like the TCP C&C flows but are not periodic). Aggregation every 30 minutes.

5.2.2 The TCP C&C Channel

At the same time that the UDP C&C channel started, the botnet also started connecting using the TCP protocol. When we created and analyzed their 4-tuples, these connections were identified as a new TCP C&C channel. This section deals with the behavior of this C&C channel.

The TCP C&C channel was composed of 22 unique *established* 4-tuples (unique IP addresses). They are few if we compared them to the 5,699 IPs of the UDP C&C channel. These 22 4-tuples lasted between 1 and 56 days, with a median of 28.5 days. Figure 5.6 shows the behavior of this channel. The flows inside each of the 4-tuples changed their *state* several times during its lifetime. That is, instead of being *established* all the time, it was usual that the flows become *attempted* or they simply timed out. In contrast with the UDP C&C channel, these changes made the TCP C&C channel more unstable and erratic. For example, the 4-tuple to IP address 155.230.189.121 and destination port 6,758 sent 63 flows during 1 day, but only 14 (22.2%) were *established* (with a periodicity of 30 minutes). The rest of the flows were not answered and become *attempted*. These changes may indicate that a TCP C&C channel might be more prone to being blocked, filtered or taken down.

The most stable 4-tuple was to IP address 84.59.151.27 and destination port 3,285, which lasted 35 days. It had 1,058 flows and 789 of them (74.5%) had a periodicity of 30 minutes.

This TCP C&C, considering all its IP addresses, was active during the whole botnet capture, with a median of 1 flows every 30 minutes. However, on special moments there were up to 48 flows every 30 minutes. Such an increase may indicate a more intense communication. It is worth noting that we found out that a botnet C&C channel is only periodic when it is idle. When it is sending or receiving information the periodicity is lost in favor of performance.

5.2.2.1 The TCP Actions

The main difference between the UDP and TCP channels was that the TCP channel did not show a strong group behavior. However, we found a new type of TCP flows that are not part of the TCP C&C channel and that we called *TCP Action flows*. They are similar to the TCP C&C 4-tuples except that they are not periodic and they usually contain a small number of flows that transfer a large number of bytes. These flows are only used on special moments that seem to be related with some botnet actions. To compare these TCP Actions with the TCP C&C channels we separated and aggregated the traffic in thirty minutes groups. Figure 5.6 shows a comparison between the number of TCP C&C flows and TCP Action flows using the thirty minutes groups.

Just like in the UDP C&C channel, we identified the top groups of the TCP Actions. If we consider the distribution of the number of flows per group, the top groups are the groups that have a number of flows that is more than two standard deviations from the mean of the distribution. On a normal distribution it is usually the top 2.4%. Although

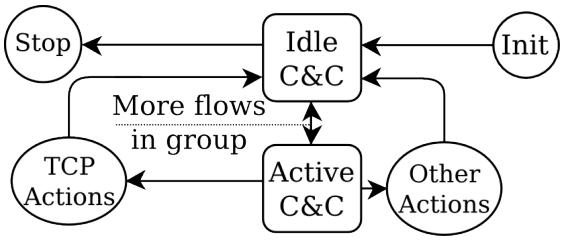

FIGURE 5.7: TCP C&C channel state model.

this is not a normal distribution, this ad-hoc limit allow us to focus on the groups that were probably representing actions in the botnet. The Figure 5.6 also shows these top groups.

From all the 2,670 groups that had at least 1 TCP C&C flow, 1,246 (46.66%) had exactly 1 TCP C&C flow, 333 (12.47%) had more than 1 TCP C&C flow, 142 (5.31%) more than 2 TCP C&C flows. Finally there were 53 (1.98%) top groups because they had more than 5 flows in them. From the 191 groups that had at least 1 Action flow, 70 (36.64%) had more than 11 and were top groups. On 26 (0.93%) groups, both the TCP C&C and Action flows had top flows, meaning that they could represent moments of important changes in the botnet.

5.2.2.2 The TCP C&C State Model

The TCP C&C channel presented some behaviors that can be modeled as states. Figure 5.7 shows this state transition model. The boxes represent states and the circles actions. The most important changes occur when the C&C channel becomes active, and when the TCP Actions are sent. The "More flows in group" threshold is the 5 flows threshold defined previously. Other actions are related with this channel, but they are discussed in Section 5.3.

5.2.3 The HTTP C&C Channel

The Zbot botnet analyzed in this chapter also started a new type of C&C channel after 32 days of operation. This new C&C channel used the HTTP protocol with encrypted payloads. The encryption was verified in the same way as with the UDP channel in Subsection 5.2.1. This HTTP C&C channel only used five different IP addresses. The characteristics of the 4-tuples of these IP addresses is shown in Table 5.2. This Figure shows that the IP addresses had a large number of periodic flows, in contrast with the TCP and UDP C&C channels that constantly lost the periodicity. This was the only C&C channel that had a periodicity different than 30 minutes.

IP	# Flows	Periodic	Periodicity (min)	Dur.	URL
95.211.9.145	2,778	99.3%	5	26d	http://95.211.9.145/m/IbQCFVVjju9O(...)
212.124.126.66	4,398	99.7%	5	24d	http://212.124.126.66/m/IbQCFVVjjl9(...)
194.28.87.64	56,841	10%	5	25d	http://194.28.87.64/sdughwejas(...)/file.php
97.74.144.110	3,682	13%	4-5	25d	http://site-serv.com/redir.php
62.149.140.209	24,529	79%	11	25d	http://www.cam-spa.net/cache/check.php

TABLE 5.2: Comparison of the characteristics of the five IP addresses in the HTTP C&C channel. All connections are to port 80/TCP.

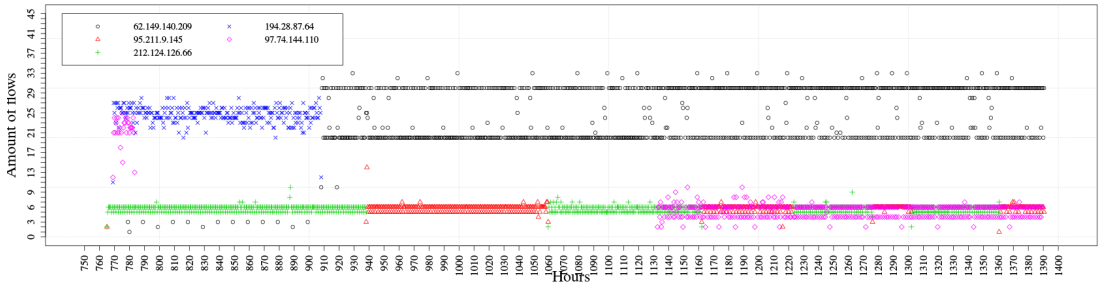

FIGURE 5.8: Comparison of the number of flows for the five IP addresses in the HTTP C&C channel.

The first two IP addresses in the Table 5.2, i.e. 95.211.9.145 and 212.124.126.66, belonged to the same C&C sub-channel. In this sub-channel the bot sent the same type of GET requests, the same type of responses and it alternately connect to them every few days. Figure 5.8 shows a comparison of the number of flows every thirty minutes for the complete HTTP C&C channel. It can be seen that the bot connected to the IP address 95.211.9.145 for the first time near the hour 765. After that, the bot switched to the IP address 212.124.126.66 during 9 days. All the responses to these two IPs contained the text "l0sM/OJnke52FQI=". Near the hour 940 the bot switched to the IP address 95.211.9.145. It sent the same GET requests every 5 minutes and got the same answer. After six days (hour 1,060) the bot switched to the IP address 212.124.126.66, this time receiving as answer the text "q+RkdjZAgJsfSj4=". Five days later (hour 1,162), the IP address 212.124.126.66 sent a new special answer of 255 characters to the bot. After this special answer the bot connected to several web pages, indicating that the new answer was probably an order. This alternating behavior continued until the end of the capture. In total, this C&C sub-channel sent 58 different GET requests to both of these IP addresses and it received 21 different responses.

The last three IP addresses shown in Table 5.2 belong to another C&C sub-channel. They served similar PHP files and get similar requests. The bot sent POST requests to

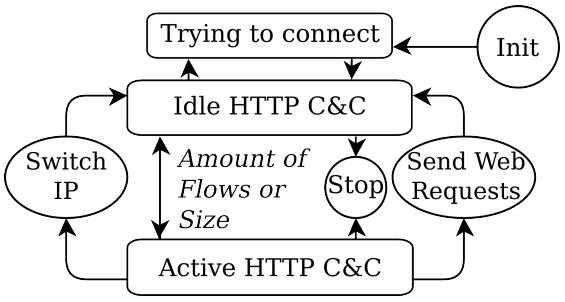

FIGURE 5.9: State model of the HTTP C&C channel.

the IP address 194.28.87.64 for 6 days (hour 910). After those six days the IP address stopped answering packets.

The HTTP connections to the IP address 97.74.144.110 were the firsts to use a host name. We could differentiate two different behaviors. First, the bot sent POST requests during 14 hours. After that, the IP address was not contacted for 15 days. Second, the bot tried to use the IP address again, but the web page was not working anymore. The answers, then, alternated between a 404 "Not Found" web page (65% of total flows) and not answering at all (15.72% of total flows).

The last IP address of the Table 5.2, 62.149.140.209, was contacted for the first time near the hour 780. On that moment, the bot downloaded a binary file from "/cache/abuild.exe" and then it received a 404 "Not Found" web page as an answer for the next five days. After that (on hour 910), the bot started receiving a 500 "Internal Server Error" answer.

5.2.3.1 The HTTP C&C State Model

The behavior of the HTTP C&C channel can be modeled similarly to the previous analysis. Figure 5.9 shows the model that represents the most important behaviors. The boxes represent states. The bot may remain on those states. The circles represent actions. The important parts of the model are the *Trying to connect* state and the *Idle* and *Active* states. The first one is responsible for most of the traffic in the channel, since it generated more flows. The last two make the difference between a working and a not working botnet.

5.3 Comparison of C&C behaviors

This section compares all the previous C&C channels to find the global behaviors of the botnet. Apart from the C&C channels, the botnet also sent a huge number of flows to the domain google.com (with and without TLS), sent large numbers of DNS flows,

accessed web sites and downloaded executable files. To analyze these actions, all the flows were aggregated into thirty minutes groups. The comparison between all the C&C channels, TCP Actions, access to google.com, DNS resquests, Web sites accesses and binary downloads can be seen in Figure 5.10. From all the C&C channels, only the UDP C&C channel is restricted to its top groups (the top 2.4% of the groups). The vertical lines indicate when a binary executable was downloaded from the web.

By correlating the information in Figure 5.10 we identified the most important behaviors of the botnet as a whole. They were usually characterized by a peak in the size of most of the C&C channels. Larger flows that deviate from the mean size of a C&C channel indicate more data being transmitted and are maybe an indicator of something different happening. These important moments are shown as numbers in the bottom of Figure 5.10. The next paragraphs analyze each of these special moments one by one.

The first moment was at hour 0, when the first group had large UDP C&C, TCP C&C, TCP Actions and DNS Established flows. From this moment on, the botnet connected to www.google.com every thirty minutes until the end of the capture, probably to know whether it had Internet access. The second moment was near hour 45, when there were large flows for DNS requests, TCP Actions, TCP C&C and UDP C&C. There were no visible changes in the traffic, so it could be related with changes in the inner state of the botnet.

The third moment, near hour 120 is important because the UDP and TCP channels stopped sending large flows after five days. From this moment both channels become more stable. The fourth moment had large TCP C&C flows, probable receiving an order, and was followed by large DNS requests and large TCP Action flows. The fifth moment showed large TCP and UDP C&C flows, then DNS requests, large TCP Actions flows and finally a peak in the requests to www.google.com. The sixth moment showed the same pattern as the fifth one.

The seventh, eight and ninth moments come 10 days after the sixth one and their actions were very important. After a large group of TCP C&C flows, there were large DNS requests, large web access flows and for the first time, three binary download from web pages. The tenth moment was one of the most important ones. It was characterized by large TCP C&C flows, DNS requests, web access and binary downloads. The actions after the binary downloads were significant since the new HTTP C&C channel was created. Those binaries therefore were a huge update to the behavior. After the new HTTP C&C started there were also huge number of DNS requests. Interesting enough, the size and number of the Google requests were also affected. However we don't know why.

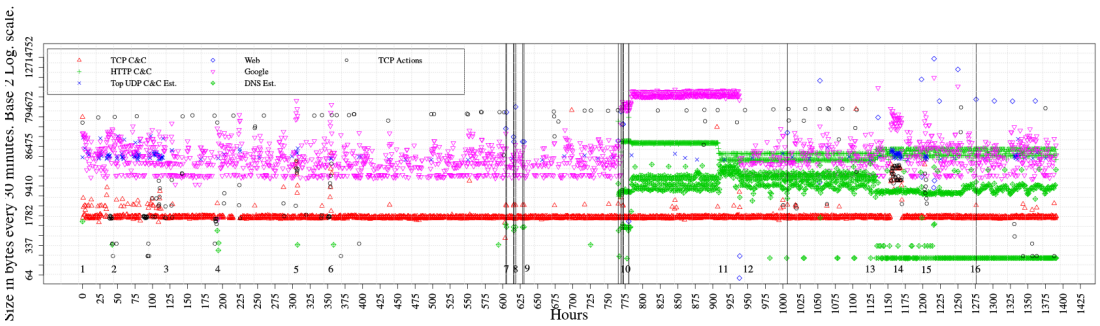

FIGURE 5.10: Comparison of the size of the flows for each botnet behavior. Size is in bytes and the scale is logarithmic in base 2. Aggregation time is 30 minutes.

At the eleventh moment there was a change in the HTTP C&C. The www.cam-spa.net domain started getting more flows and the HTTP C&C of IP 194.28.87.64 stopped working. That is why it can be seen a huge drop in the size of flows. The twelve moment is significant because after some TCP C&C flows there are fewer and smaller flows to google.com. The thirteen moment came seven days later and its most important action was to start a new HTTP C&C to IP 97.74.144.110. This generated a lot of new DNS requests that can be seen at the bottom of the Figure 5.9. On the fourteen moment and during 5 hours the TCP and UDP C&C channels and the TCP Actions sent large flows. The fifteen moment is very similar to the previous one, except that there is only a peak in the UDP C&C, the TCP Actions and the Google requests. The sixteen moment was characterized by a peak in the TCP C&C channel along with a binary web download.

5.3.1 Botnet state model

The detailed analysis of all the botnet actions in the previous Subsection was helpful to describe new relationships between the C&C channels and the actions. The identified moments where the C&C channels correlate and act simultaneously were also characterized by specific actions in the network. It is not possible to determine which C&C channel was responsible for the actions, but it is possible to identify those moments precisely. With this information, a general state model for the botnet as a whole was created. The model is shown in Figure 5.11. The states of this model were designed to be simple and to capture the most general transitions of the botnet. The model shows that a C&C channel can be Idle or can become active and start doing actions. This basic separation is an important advance in the general modeling of the states of a botnet.

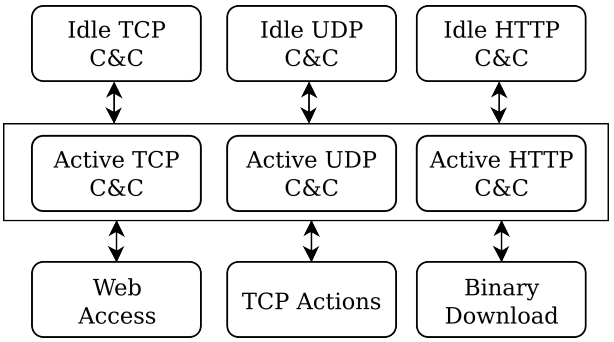

FIGURE 5.11: Botnet general state model.

5.4 Conclusion of the Behavioral Analysis

This Chapter was dedicated to the study of the behaviors of the Commands and Control channels of a specific botnet. The malware was possible a variant of the Zeus botnet and was captured in the Malware Capture Facility Project of the CTU University. The dataset was freely published and can be downloaded from the website [2].

The key highlights of this Chapter are the identification of the UDP, TCP and HTTP C&C channels in a 57-day long botnet dataset, the behavioral analysis of their flows and the creation of the state models that represents their actions. The behavioral analysis was done using 4-tuples, which were characterized by the duration of the flows, the sizes of the flows and the time differences of the flows. These characteristics help us discover when the channels were idle or active, and which type of actions were related to them. The behaviors of the C&C channels were compared to identify the complete decisions of the botnet as a whole. A basic state model was also created for the complete botnet behavior. We believe that this analysis is an incremental step in the modeling of malware behavior and can help understand the actions of the botnets better. The next Chapter describes how the models learned during this analysis are used to create a new detection algorithm.

[2] https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-25/

Chapter 6

The CCDetector Method: Botnet Behavioral Models and Markov Chains

6.1 Introduction

The deep analysis of the C&C channel behaviors made in the previous Chapter 5 showed that it is possible to create models for these type of behaviors that represents their actions. The most important insight was that C&C channels seem to change from one state to another. Based on this ideas we developed a novel state-based behavioral model of network traffic and a novel detection method called CCDetector. The CCDetector method is based on this idea that the behavior of the connections of the botnets changes from one state to another. This means that, for example, a connection that was in a periodic state sending large amounts of information can change to a non-periodic state sending a small amount of information. This idea of identifying the states of the connections over time and studying how they change is a core part of the CCDetector method.

One of the biggest differences between the CCDetector methods and our previous methods is that the CCDetector method works with individual connections. This means that instead of analyzing the complete traffic of an infected computer as a whole, we separate each individual connection from each IP address and we treat them as completely independent. In this way it may be possible to find out when an infected computer is doing some connections from the malware and some connections from the normal applications. It may be also possible to distinguish the different types of

connections from the same malware, such as attacks, C&C channels, attempts, etc. The CCDetector method focuses on the connections from C&C channels. The connections are represented as 4-tuples, a type of structure described in Subsection 6.2.2.

To analyze the behavior of individual connections in the CCDetector method, it was necessary to individually separate each connection. More importantly, it was necessary to label each connection separately. This manual process was time consuming and prone to errors, since we have to assign labels such as *From-Botnet-V1-TCP-CC102-HTTP-Custom-Encryption*, which means a connection coming from a botnet using the TCP and HTTP protocol, and using a custom encryption. This labeling process is an important part of the development of the CCDetector method. The dataset chosen to be labeled with these detailed labels was the same that was used for the BClus method. We used the same dataset because it is large, with three types of traffic and because it was used in the previous comparison with the third-party methods. The detailed labeling process is described in Subsection 8.5.

Once all the connections in the dataset are labeled, the CCDetector method groups together all the flows belonging to each C&C channel. For each flow in a C&C channel the method extracts three features that help modeling the changes over time. The features are the *size* of the flow, the *duration* of the flow and the *periodicity* of the flow. The features are motivated by our previous analysis in Chapter 5, that found when a C&C channel is active, idle, down, working or not working. Using thresholds to separate the values, these three features can identify 36 different states (described in Subsection 6.2.3) that each flow can be assigned.

Each connection, including the C&C channels, are usually composed of several flows. As each flow is assigned a specific state, each connection is represented as a chain of states, one for each arriving flow. This chain of states represents the behavior of the connection over time. Based on this chain of states we created our own behavioral state model.

After the CCDetector method created the behavioral models of verified C&C channels, it can be used as a detector of unknown behaviors. The method uses the chain of states of each known connection to create a Markov Chain (MC) model for that connection. During the detection phase, the MC models are general enough to match similar behaviors in the unseen traffic.

The results obtained were promising. Just as happened with our previous method BClus, the best way of evaluating the results is through a proper comparison. In the next Chapter 7 we describe a comparison of the CCDetector method with our previous BClus method and the two third-party detection methods.

The CCDetector method presents the following advances:

- A new state-based model for representing the network behavior of botnets actions.
- A new botnet detection method based on Markov Chains.
- A deeper understanding of the botnet actions in the dataset. Resulting in a new set of more than 1,470 different and unique labels.

The CCDetector method does not depend on the payload of the packets (protecting the privacy and even working with encrypted traffic), it does not need any a priori information about the botnet structure and it can detect single bots. Moreover, since it does not analyze the behaviors of flows but the behavior of botnet actions it is able to model actions such as ”To send SPAM”, ”To communicate with a C&C server”, or ”To download a binary file”. The next Subsections describe each part of the method.

6.2 Command and Control Behavior Detection Model

Our experience analyzing the behavior of the botnets in the network taught us that it is important to understand what the botnets are doing to create better detection models [9]. We conducted, in Chapter 8, a deep analysis of the traffic in our dataset to understand how the Command and Control channels behave and which may be good features to detect them [92]. The analysis concluded that the C&C channels could be modeled as a continuously evolving entity with different and predefined states. Each of these states represent a moment in the behavior of the C&C channel.

Since the C&C channels are continuously changing from one state to the next state, we can see all these states as a chain. The chain of states stores the historical transitions and represents the behavioral pattern of the connection. Since our method receives network flows as input data, each new flow represents a change into a new state. The states are only defined by three features that are paramount to our model: the size of the flow, the duration of the flow and the periodicity of the flow. Using these three features we are able to create 36 different states. For simplicity each state is represented and stored as an ASCII letter.

Our method operates over this chain of states as a stochastic process with the Markov Property, that is, it creates a Markov Chain that models and represents the behavior of the C&C channel. The use of Markov Chains is motivated by the transition from one state to the other and the probabilities of that transition. The Markov Chains can be generated and used later to detect unknown connections in other networks. The

creation of the Markov Chain behavioral models is done during the training phase of the method.

Once that the behavioral models are trained, they are used to test the detection method on the testing dataset. The next Subsections describe the creation of the Markov Chain model, the training and the testing steps.

6.2.1 Network Flows as Input of our Method

Most network-based detection methods use one of two types of input data for their algorithms: the packets or the flows. The packets (usually captured using a binary format such as libpcap [75]) have the advantage of including the complete payload and therefore allow researchers to analyze every byte being sent or received. There are three disadvantages on using packets. First, that they are difficult to share because of the privacy issues. Second, that it is not possible to store the traffic for long periods of time due to the high storage space required. Third, that it is difficult to mix and combine packets from different sources.

On the other side, the flow-based representations of network traffic, such as NetFlow [93], IPFIX [94] or Argus [90], aggregate the traffic based on the source IP address, source port, destination IP address, destination port and protocol (some standards may use other keys for aggregation). The aggregated data summarize the information about the flow, such as the amount of transferred bytes or the duration. The main advantages of flows are: First, that they automatically group the related packets in a connection. Second, that they drastically reduce the amount of information stored. Third, that the flows from different capture points in the network can be easily combined. The main disadvantages are that most NetFlow standards do not store the payload and that the data that is not summarized is lost.

As it was stated in Subsection 5.2, our detection method uses the flow definition of the Argus suite. This format was selected because of the following advantages: it generates bidirectional flows, it stores the flows in a binary file, it includes tools to process the flows, to apply labels, to filter flows, to plot flows, and it can be fed with live or offline pcap packets. Although it should be possible to use uni-directional flows in our model, the bidirectional flows solve the dilemma of finding which are the server and the client. The only difference between the output flows of our model and the NetFlow standard is that our flows are bidirectional.

Any type of flow representation is limited to the information that it can summarize. In the case of NetFlow compliant standards they usually include information about

the interfaces, timestamp, duration, bytes, packets, source IP address, source port, destination IP address, destination port, protocol, type of service and flags. For our detection method we process each Argus flow and we extract both IP addresses and both ports, the start timestamp of the flow, the size of the flow and the duration of the flow. The next subsection describes how this information is pre-processed to be used in our method.

6.2.2 Preprocessing of Flows: 4-tuples and Features

Despite the useful information usually stored in the network flows, we found that it is not enough to represent the continuously changing state of the C&C channels. Our deep analysis of the botnet traffic concluded that a C&C channel with a single and identifiable behavior is a connection coming from a bot to a specific *destination service* in a specific *destination server*. Therefore, to find out and work with each of these individual behaviors, we created a new structure called 4-tuples that aggregate all the flows coming from a bot to a specific destination port and destination IP address. In this way, ignoring the source port of the connections, it is possible to further aggregate the flows that share the same source IP address, destination IP address, destination port and protocol.

One of the reasons of why the 4-tuples are needed is that each new connection in a C&C channel uses a new source port. Therefore, each new connection generates a new flow in the network, since each flow has a different source port. Consequently, it is not possible to analyze the periodicity of a C&C channel by analyzing only *one* flow, because the channel contain many flows. The 4-tuple structure is capable of capturing all the flows of the C&C channel.

In the CCDetector method, each network behavior its represented by a 4-tuple, and each 4-tuple stores the state of that network behavior by aggregating the information coming from its flows. In the aggregation process, each arriving flow is assigned to its corresponding 4-tuple and three features of the flow are extracted: its **size**, its **duration** and its **periodicity**. The size and duration of the flow are directly taken from the flow data, but the periodicity is computed from the timestamp of the flow. The three features are used to select which state should be assigned to each flow. The procedure to compute the periodicity is described in the next Subsection.

6.2.2.1 Periodicity Analysis

The periodicity of botnet traffic has been analyzed and for detection before [95][85][6]. It has been reported as a good feature because most automatic actions from the botnet, specially the C&C channels, were found to be periodic. The periodicity of the C&C channels may be explained from a simple algorithmic point of view: If you need a regular event, the simplest way to achieve it may be a loop with a sleep function. It is not impossible for the botmasters to have non-periodic C&C channels, but so far none has been reported in the wild. The reasons for this may be that non-periodic C&C channels do not allow for simultaneous and coordinated actions.

After analyzing the captures in the MCFP [89], we found that there were two aspects of the periodic C&C channels that were not thoroughly explored. The first one is that the periodicity is never *perfect*. The second one is that the same botnet, and even the same C&C channel usually show several different periodicities during its life-time.

The assertion that the periodicity is not perfect is explained if we consider that there are several unknown factors affecting any connection, such as network lag, network delay, congestion, process of orders, etc. They add noise to the timings of the C&C packets. To illustrate this, we analyzed the MCFP capture called CVUT-Malware-Capture-Botnet-25 that is publicly available [1]. In this capture, an example of the variance in the periodicity of a connection can be found in the 4-tuple *10.0.2.106-212.124.126.66-80-tcp* that was recognized and labeled as an HTTP C&C channel. The manual analysis of the packets of this 4-tuple [92] shows that it is an encrypted and periodic HTTP C&C channel sending URLs such as *http://212.124.126.66/m/IbQCFVVjjl9Ob3XaHe(...)*. An analysis of the variance of the periodicity is shown in Table 6.1. In the Table, T2 refers to the time difference between two consecutive flows. It can be seen that the periodicity of the flows is around 5.3 minutes, but it is not precise. A deeper analysis of the periodicity values for the 3,940 flows in this 4-tuple shows that the minimum periodicity value was 0.0 sec, the 1st Quartile was 317.1 sec, the Median 319.3 sec, the Mean 544.2 sec, the 3rd Quartile 325.8 sec and the maximum value was 437,400 sec. The mean was slightly higher than the median because at some point the C&C channel was idle for 5 days until it started again.

Table 6.1 is one of many examples of C&C channels having a defined but slightly unstable periodicity. In this case the C&C was idle. Other C&C channels are down or are active and therefore have different periodicities [92].

Another useful example of periodicity in a C&C channel is shown in Table 6.2. It shows the time differences for the 4-tuple *10.0.2.106-89.40.177.36-2670-tcp* in the same

[1] http://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-25/

Start Time	T2 (secs)	Bytes	Duration (secs)
22:34:21.956655	317.492613	1,074	1.285
22:39:39.540718	317.584063	1,074	1.629
22:44:57.477837	317.937119	1,074	0.789
22:50:14.573753	317.095916	1,074	0.871
22:55:31.739394	317.165641	1,074	1.029

TABLE 6.1: Example periodicity for the flows of tuple 10.0.2.106-212.124.126.66-80-tcp in dataset CVUT-MALWARE-CAPTURE-BOTNET-25. T2 is the difference between the current flow and the previous flow. This is a stable and idle C&C channel.

Start Time	T2 (secs)	Bytes	Duration (secs)
08:39:01.415443	48,500.387487	186	8.998
20:11:25.206581	41,543.791138	1,585	0.610
20:41:25.820231	1,800.61365	1,580	0.587
21:11:26.409415	1,800.589184	1,610	0.549
21:41:26.958067	1,800.548652	1,513	0.575

TABLE 6.2: Example periodicity for the flows of tuple 10.0.2.106-89.40.177.36-2670-tcp in dataset CVUT-MALWARE-CAPTURE-BOTNET-25. T2 is the difference between the current flow and the previous flow. This is a not so stable C&C channel with a larger periodicity.

dataset CVUT-Malware-Capture-Botnet-25. This 4-tuple was slightly unstable, perhaps because of being active and performing actions in the network. When the channel was idle it had a periodicity of approximately 30 minutes.

Since the focus of our approach is to detect the behavior of the C&C channels, these previously analyzed time differences raise an important question: How can we capture the periodicity of C&C channels in such a way that, independently of the variances of the frequencies, it can still be used to detect the C&C? The solution used in our model is to compute the second-level time differences of the flows, in order to search for *how much* periodic these connections were. The usual approach for dealing with periodicity [95][96] has been to separate the traffic into bins and find if the same connection appears every certain amount of bins. However, our approach to compute the periodicity is completely new and therefore capture other properties of the traffic.

To compute the periodicity, the CCDetector method first computes the time difference between the first flow and the second flow, called T2. Then, it computes the time difference between the second flow and the third flow, called T1. Finally, it computes $TD = T2 - T1$. Table 6.3, shows the TD values for the 4-tuple *10.0.2.106-212.124.126.66-80-tcp*. It can be seen that when the flows are strongly periodic the values of TD are near 0. Table 6.4 describes the TD values for the 4-tuple *10.0.2.106-89.40.177.36-2670-tcp*. It that can be seen that when the C&C is not periodic the TD values are greater than 0.

Start Time	T2 (secs)	Bytes	Duration (secs)	TD (sec)
22:34:21.956655	317.492613	1,074	1.285	0.5
22:39:39.540718	317.584063	1,074	1.629	0.1
22:44:57.477837	317.937119	1,074	0.789	0.4
22:50:14.573753	317.095916	1,074	0.871	-0.8
22:55:31.739394	317.165641	1,074	1.029	0.1

TABLE 6.3: Example values of the second-level time difference TD for the 4-tuple 10.0.2.106-212.124.126.66-80-tcp. The strong periodicity produces TD values near 0.

Start Time	T2 (secs)	Bytes	Duration (secs)	TD (sec)
08:39:01.415443	48,500.387487	186	8.998	48,352.7
20:11:25.206581	41,543.791138	1,585	0.610	-6,956.6
20:41:25.820231	1,800.61365	1,580	0.587	-39,743.2
21:11:26.409415	1,800.589184	1,610	0.549	-0.02
21:41:26.958067	1,800.548652	1,513	0.575	-0.04

TABLE 6.4: Example values of the second-level time difference TD for the 4-tuple 10.0.2.106-89.40.177.36-2670-tcp. An unstable periodicity that produces some values larger than 0.

Table 6.4 also shows that when the C&C is periodic again (independently of the value of the frequency), the values of TD are near 0 again. This shows that our periodicity feature TD is independent from the actual value of the frequency.

Another motivation to create our new periodicity feature is the fact that a botnet tend to have different frequencies on its C&C channels. Moreover, each C&C channel in the same botnet tend to have a different frequency during its life time. A good example of these variances can by seen in the Sality botnet that was captured in our MCFP [2] and is shown in Figure 6.1. In this botnet there are at least three different periodic connections using three different frequencies. The first periodic C&C channel corresponds to TCP Web connections that are done every 12 hours. They can be seen as five large blue peaks in the Figure. The second periodic C&C channel is also composed of TCP Web flows but with an approximate frequency of 1 hour. It can be seen in all the blue small peaks. The third periodicity is on a C&C channel with only TCP flows and a frequency of approximately 40 minutes. All of them are C&C channels in the same botnet.

[2] http://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-25

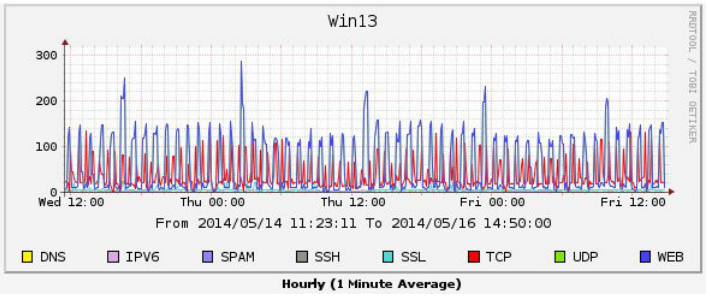

FIGURE 6.1: Sality Botnet showing three different frequencies in three different C&C channels. A first frequency of 12 hours in a Web-based C&C channel, a second frequency of 1 hour in another Web-based C&C channel and a third frequency in a TCP-based C&C channel.

6.2.3 Behavioral State Model. The Core of the CCDetector Method

The most important characteristic of our proposed detection model is the relationship of each flow with a state change in the botnet behavior. We hypothesize that the behavior of the botnet can be viewed as a state-machine that changes its state upon the arrival of a new flow. This idea was born from the way in which botnets are used. The botmaster may keep the botnet in an idle state for some time and then decide to change its state to send SPAM or to update itself, to finally put it back in an idle state [92]. The main idea of the state model is to find out these changes and to store its behavior.

In our model, we create the 4-tuples (as it was described in Subsection 6.2.2), and therefore we separate each connection of the botnet that access a different service. The creation of the 4-tuple means that we are actually separating each independent behavior of the botnet and we are generating a model for each of them. This is important from the point of view that we are not generating a unique behavioral model for the botnet, but instead we are generating a new behavioral model for each botnet *connection*. This approach allow us to concentrate on individual connections to label them correctly and to use these models later.

For each arriving flow from the network we extract its corresponding 4-tuple. Upon arrival, each flow is assigned a state and this state is added to the corresponding 4-tuple. To decide which should be the state being assigned we look at our three previously defined features: its *size*, its *duration* and its *periodicity* of the flow. Using two thresholds for each feature (six thresholds in total), we separate the values of each feature in three groups. In the case of the size we can say that the flows are *small*, *medium* or *big*. In the case of the duration we can say that the flows are *short*, *medium* or *long*. And in the case of the periodicity we classify them as *Strongly Periodic*, *Weakly Periodic*, *Not Periodic* or that there is *no enough data* to decide.

	Small Size			Medium Size			Big Size		
Duration	Short	Medium	Long	Short	Medium	Long	Short	Medium	Long
Not enough data	1	2	3	4	5	6	7	8	9
Strongly periodic	a	b	c	d	e	f	g	h	i
Weakly periodic	A	B	C	D	E	F	G	H	I
Not periodic	r	s	t	u	v	w	x	y	z

FIGURE 6.2: The 36 different states that each flow can take when assigned to its 4-tuple.

These six thresholds allow us to assign to each flow one of 36 different states. For visualization purposes, each state is represented by a letter. This is important because using letters allow a human expert to understand and verify the method easily. Figure 6.2 shows the assignment logic of letters to each state. Each arriving flow is assigned one of these letters and the concatenation of all the letters is the state model of the 4-tuple. There is only one more letter that is not in the Table, the letter 0. This means that a flow has timed out. The time out marks a connection that has probably started again. As far as we know this is a new type of assignment of states to flows that opens new opportunities for the modeling of network behaviors.

To be useful, this logic of states assignment must be used with good thresholds. The success of the model depends on this selection. The next Subsection describes how these thresholds were selected using our dataset.

6.2.3.1 Defining the thresholds of the state model

To fix the threshold values of the state model of our method means to find which are the values of the three features that maximize the separation of flows for the C&C channels. The idea of using thresholds when assigning states is to roughly identify the changes in the behaviors of the C&C. The values for the thresholds should only come from the analysis of the C&C channels and not from all the botnet traffic because the focus of our method is to detect C&C channels.

To select the thresholds we should first get all the relevant data from the training dataset. In Subsection 8.5 there is a description of the labeling process along with a description of the separation of the dataset into training and testing. The thresholds should be trained only with the training dataset. The basic idea is that, for each feature, its first threshold should separate the first 33% of the dataset and its second threshold should separate the second 33% of the dataset. Table 6.5 shows the basic statistical characteristics for the three features computed from the botnet C&C 4-tuples of the training captures in the dataset. The table shows that the C&C flows have a median TD of 1.97, which

Feature	Min	1st Qu.	Median	Mean	3rd Qu.	Max
size	184	3,505	12,287	45,512	47,195	615,599
duration	0.0002	2.84	5.03	14.56	11.01	254.96
TD	0	0.99	1.97	840.11	4.66	11,3558.65

TABLE 6.5: Statistical characteristics of the three features for the Botnet CC flows in the training captures.

means that they usually have a good periodicity. It can also be seen that the median duration of C&C flows is 5 seconds, which is a clear contrast with the normal data we analyzed, which usually have longer flows. Finally, if we consider that one packet alone can transfer up to 1,500 bytes, the size of the C&C flows is usually small, because the median size for the flows is only 12,287.

To train the correct values for the thresholds, and based on Table 6.5, we computed the ECDF (Empirical Cumulative Distribution Function)[3] of each of the three features in the training dataset and then we found out the values of the features for the 33% and 66% of its distribution function.

In the case of the feature *size*, the 33% of the ECDF function has a value of 6,092.16 bytes, and the 66% has a value of 25,841.8 bytes. In the case of the duration the 33% of the ECDF function has a value of 3.310138 seconds and for the 66% it has a value of 7.650338 seconds. In the case of the TD the 33% of the ECDF function has a value of 1.436069 seconds and for the 66% it has a value of 3.401405 seconds.

Finally we also defined the value for the time_out of our logic. It was defined as 3,600 secs (one hour) because that is the timeout for connections in Argus suite and other NetFlow standards.

The final thresholds selected for our model are:

- Size threshold 1 = 6,092 bytes (6092.16)
- Size threshold 2 = 25,841 bytes (25841.8)
- Duration threshold 1 = 3.3 seconds (3.310138)
- Duration threshold 2 = 7.6 seconds (7.650338)
- TD threshold 1 = 1.4 seconds (1.436069)
- TD threshold 2 = 3.4 seconds (3.401405)
- Timeout threshold = 3,600 seconds

[3] https://en.wikipedia.org/wiki/Empirical_distribution_function

Once that the thresholds had been fixed, it is possible now to use the state generation logic to produce the states of each 4-tuple. After applying this logic to each new flow arriving to a 4-tuple, the tuple can be seen as a chain of letters that represent its behavior. This complete set of letters, or chain of states, is the base of our detection method.

For example, the chain of state for the 4-tuple *10.0.2.106-212.124.126.66-80-tcp*, that was previously analyzed in Table 6.3 and that has the label *From-Botnet-V1-TCP-CC102-HTTP-Custom-Encryption* is the following:

```
11aaaaaaaaaabrrctrraaaAaaaaaAaaaaaaaaaaaaaaaaaAAAaaaaaaaaaaaaaaaaaaAaAa
aaaaaaaaaaaaaaaaaaaaaaAAaaaaaaaaaAAaaaaaaaaaaaaaaaaaaaaaaaaAAAAaaaaaaaaAAa
aAAaaaaaaaaaaaaaaaaaaaaAAaaaaaaaaaaAAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAaAaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAaaaaaaaa
aaaaaaaaaaaaAAaaaaaaaaaAaaAAAAAAAAaaaaaaaaaaaaaaaaaaaaaaaaaaaa(...)
```

According to the assignment logic it can be seen that this 4-tuple has a good periodicity because the flows are being assigned the letter *a*. From time to time the flows have a weaker periodicity (letters *A*), and sometimes it looses the periodicity for some flows (letters *r* and *t*).

In contrast, the 4-tuple *10.0.2.106-195.113.214.215-80-tcp* that has the label *From-Botnet-V1-TCP-HTTP-Google-Net-Established-1* was assigned the following chain of states:

```
71r0rxrurradrrAAarraArudaurrDuradrrDdrraadurrgdruAxrruruDAxrDuarxraaduua
durrrruurdruurrG0rurarruurur0rur0rruuurrrudrurraruadurrudruuaruxurarrrur
aurururrruAdrrruu0rrrurrraruaarrruraauurrrarrrrrurrurrurrurrrrrrrxuurrur
rrrrurruuuurrrrrrruurrrarrauuarrarx0rruruaarruurrrrauAaduuddruaaruuaaaru
ruarrDrrdruDrraaarrrDddxrrrdaxraDdurrrrurrrrurrrrrrrrrurDrr(...)
```

This chain of states represents a typical connection to Google services and is not related with any C&C channel. It can be seen that it does not have a very periodic behavior (although there are periodic flows from time to time with letters *a*, *A*, *d* and *D*). Most of the flows are letter *r*, *u* and *x*, which means non-periodic flows with variable sizes. There are also many *0* letters, meaning that the time difference between the flows were more than one hour and therefore the connection timed out.

Another example of a Google connection is the 4-tuple *10.0.2.106-195.113.214.211-80-tcp* that was assigned the label *From-Botnet-V1-TCP-HTTP-Google-Net-Established-1* and has the following chain of states:

```
11r0rrrrrrrrarrrrr0t0rraaaaaa0rraarrrraaaraaa0rrarArraaarrarArrrraaa0r
raaarraaarraaarrraaa0rraaarraaarraaarraaa0rraaarrraaarraaarrrrr0rraaar
rraaarraaarraaarraaarrraaarraaarraaa0rraaArrraaarraaarraaa0rraaarraaar
raaarraaarraaarraaarraaarraaa0rrrAAArrAaarrAAa0rraaarrrrAar(...)
```

This chain of states has some periodic flows with letters *a* and *A* that can last up to six flows and are separated by moments of non-periodic flows with the letter *r*. There are also some time outs of more than one hour (letters *0*).

The previous chains of states are good examples of the remark done on Section 7.1 about the complexity of the behaviors. It is not possible to say if the Google connections are periodic or not. It is only possible to say that sometimes they show a periodic behavior and sometimes they show a non-periodic behavior. Using this state model it may be possible to identify the relationships between the periodic states and to differentiate the connections.

6.2.4 Markov Chain Model for Representation and Detection of the Behavior of C&C Channels

After assigning the letters to each flow using the logic described in the previous Subsection, each 4-tuple has a chain of states that represents its behavior. These chains also encode an important information about the botnet connection: the transitions between states. How each C&C channel changes from state to state its an important part of its behavior. We used Markov Chains to model this behavior, based on the transitions between states.

A Markov Chain is a mathematical system that models transitions from one state to another on a state space [97]. Usually the transitions are *memory-less*, meaning that the transition probability only depends on the current state and not on any other previous state. This is called the Markov property [58]. In our model, this property is not completely fulfilled since our feature called *TD* is computed using values from the two previous flows. However, we believe that a first order Markov Chain may still be a good choice for our method.

Our method uses Markov Chains for two purposes: First, to model the behavior of each C&C channel (during training). Second, to detect, in unknown networks, similar behaviors as the already known C&C models generated on the training. Both purposes are explained in the following Subsections.

6.2.4.1 Modeling the Behavior of the C&C Channels of Botnets with Markov Chains

The first use of Markov Chains in our method is to model the behavior of the C&C channels in the training dataset. The idea is to create a Markov Chain using the chain of states from the C&C channel. However, there is one issue to solve regarding the labels and 4-tuples. Each C&C channel only has one label. The label is even used to identify the C&C channel. The issue arises when a C&C channel is composed of several 4-tuples. This may happen when a channel is related with different connections, for example when the C&C server uses a domain name but the IP address of the domain name changes. In this situation all the 4-tuples belong to the C&C channel. Since each 4-tuple has its own chain of states, the C&C channel has to deal with multiple chain of states. To solve this issue we added the letter #, which is meant as a chain separator. All the chain of states of all the 4-tuple for each C&C channel are concatenated and separated with this letter. In this way we can create a unique Markov Chain for the channel by using this letter as a separator. The library software that we used, called PyKov [4] uses this letter to create the model correctly.

Apart from the Markov Chain, the other important information that is stored for each C&C in the model is the probability of generating the original chain of states using its own Markov Chain. This value is later used in the detection phase. However, since the chain of states for each label consist of the concatenation of several chain of states (all the 4-tuples having the same label), we use the longest unconcatenated chain of states for computing this *original* probability.

The training phase consists in generating these models for each ground-truth label in the training dataset. The complete model that is generated and remembered is composed of the following components:

- Its probability matrix (part of the Markov Chain).
- Its initialization vector (part of the Markov Chain).
- The probability of generating the original chain of states using its own Markov Chain.

6.2.4.2 Markov Chains for Detecting the Behavior of the C&C Channels of Botnets

The second use of Markov Chains is during testing, for detecting similar behaviors in unknown network traffic. The detection phase consists in extracting the bidirectional

[4] https://riccardoscalco.github.io/Pykov/

Argus flows from the testing traffic, creating the 4-tuples and computing the chain of states of each 4-tuple. After this, each chain of states is compared with each trained model during the training phase. Based on this comparison the label of the winning model is assigned to the unknown 4-tuple. Since each 4-tuple is composed of several flows, each flow can be assigned the predicted labeled.

For each unknown chain of states in the testing dataset, the main idea of the detection phase is to find out which trained Markov Chain model probably generated it. The usual process should be to calculate this probability for each trained Markov Chain and then to select the one with the higher probability. However, there is a shortcoming. It is very likely that the unknown 4-tuple being analyzed has some states in the chain that are not part of the Markov Chain used for the calculation of the probability. For example, we may have a trained Markov Chain that was generated with the chain of states "81aaababababa" and we may have an unknown 4-tuple which chain of states to be tested is "11zzzzzzzzyyy" (testing chain of states). If we compute the probability of the testing chain of states of being generated by the trained Markov Chain it would be zero. This result is probably correct given how different the chains of states are.

However, this also may affect the generalization of our method sine the traffic from a C&C channel in one botnet is usually slightly different from the C&C traffic in another botnet. For example, we may have a trained Markov Chain that was generated with the chain of states "81aaababababa" and we may have a testing 4-tuple which chain of states is "81aaaabaababafxzfzfzzzz" (called from now on *testing chain of states*). This situation is common on C&C channels that stop working ('xzfzfzzzz' letters) after successfully working for some time ('81aaaabaababa' letters). The probability that this trained Markov Chain generated the *testing chain of states* is also zero because some transitions are not in the trained Markov Chain. However, it is clear that part of the *testing chain of states is* really similar to the original chain of states and may be generated by the same C&C channel. To solve this issue, we designed a detection method that can compare the chain of states from a 4-tuple with a Markov Chain that does not contain all of its states. The process can be summarized as follows:

1. For each unknown testing 4-tuple:
   (a) Compute its *testing* chain of states.
   (b) For each candidate model (Markov Chain):
      i. Get the *original* probability of generating the *original* chain of states with this Markov Chain (included in the trained model).
      ii. Get the OSI 3rd layer protocol of the *testing* 4-tuple: could be one of TCP, UDP, ICMP, ARP or not defined.

iii. If the *testing* 4-tuple protocol is defined and it matches the protocol of the candidate model, then continue.

iv. If the *testing* 4-tuple protocol is defined and it does not matches the protocol of the candidate model, stop the comparison and discard this model as candidate.

v. If the *testing* 4-tuple protocol is not defined, then continue.

vi. Compute the probability of the *testing* chain of states using the candidate model. If one transition of the chain of states is not in the candidate Markov Chain model, ignore the transition and move to the next letter.

vii. If the absolute difference between the *testing* probability and the *original* probability is greater than a certain probability threshold, the model is discarded as a candidate.

viii. Store the *testing* probability along with the name of the candidate model.

(c) Select the candidate model with the lowest probability as the final model.

The decision to compare the testing chain of states against the model only when the 3rd layer protocols are the same (steps `iii`, `iv` and `v`) was taken based on our experience analyzing the training data. We found that the behaviors of these protocols can be similar even when they are completely unrelated. For example, ICMP packets sent by the ping utility have a strong periodicity. To better discern between these behaviors we only compare them if they use the same protocol. This decision does not mean that we discard some protocols, but that we only compare between the same protocols. If the protocol is unknown, then we compare the testing chain of states against all the candidate models.

A very important part of the method is step `vii`, where the absolute difference between the *testing* probability and the *original* probability is computed. The main motivation for this comparison is that we want to find out the 4-tuples which behavior is similar to the original C&C behavior. This usually means that the probability of generating both chains of states should be similar. To be similar means that the difference between both probabilities is not large. The final decision to know if the probability is similar enough is our *probability threshold*, which let us tune how similar the chains must be to be considered the same behavior. The probability threshold is important because if it is not used, then there will be *always* some model that will be selected as winner and therefore assigned to the testing 4-tuple. This threshold is a way to decide that a testing 4-tuple is unknown.

Finally, all the candidate models that were selected are sorted according to their absolute difference and the one with the lower difference is selected.

6.2.5 Training Methodology for the Detection Method

This SubSection describes the training methodology of the CCDetector method. There are two main models in this method: First, the states model that assigns letters to each flow according to a logic and generates a chain of states. Second, the Markov Chain model that computes the transition probabilities from these chains of states and creates a behavioral representation that is used for detection. Both models need to be trained. The thresholds of the states model were already defined in 6.2.3.1. This subsection describes the general training of the CCDetector method and specifically the training of the Markov Chains model.

The generic training methodology consist of the following steps:

1. Read the bidirectional flows from the Argus file.
2. Separate the 4-tuples.
3. Assign the manual ground-truth labels to each 4-tuple in the training dataset.
4. For each 4-tuple which labels says that it is a *C&C channel*, generate a chain of states according to the logic described in Subsection 6.2.3.
5. Determine the thresholds of the state model as described in Subsection 6.2.3.1.
6. Merge together the chains of states of all the 4-tuples having the same C&C label. The chains are separated with the special character #.
7. Generate the Markov Chain for each label.
8. Store the behavioral models on disk. A behavioral model includes the Markov Chain (probability matrix and initialization vector), the probability of generating the original state chain with its own Markov Chain and the label.
9. Preselect the best Markov Chain models based on the length of its chain of states. This means that if the chain of states is too short, we discard it as a candidate.
10. Train the probability threshold on the training-validation dataset.

The assignment of ground-truth labels to the dataset is a manual process that requires a deep understanding of the network traffic generated by the botnet. This expert process is the only way to correctly label our dataset and therefore to correctly verify our method and completely describe our results. The technical description of the labeling process, i.e. rules and programs, is described in Subsection 8.5.1, but here we will assert its importance. A correct label assignment process is paramount for a good evaluation of

any detection method. It is so important that [98] proved that if the training data is mis-labeled, some boosting algorithms fail to produce a model with accuracy better than 50%. In particular they concluded that "convex potential boosters cannot withstand random classification noise". This gives an idea of how a mis-labeled dataset could affect a classification algorithm.

Two clarifications should be made about the step 7. First, the models for the Background labels are not generated. Second, no Markov Chain is generated for a chain of states which length is less than three letters (i.e. three flows). This limit is part of the design of our state model since to compute the *TD* value (time difference of time differences), at least three flows are needed. Therefore, no 4-tuple can be generated with less than three flows.

The last step of the training methodology is to find out the probability threshold. The description of how this threshold is used is on SubSection 6.2.4.2. To find out the best probability threshold we execute our method on the training-validation dataset with different threshold values. The behavioral models used are the ones created with the training-validation dataset. In this way, the selection of the threshold is not influenced by the testing dataset.

6.2.6 Testing Methodology of the Detection Method

The training phase of our detection method created several detection models from the training dataset. These models can be used now in the testing phase to find out the real performance of our method. This phase runs the method with the trained models on the testing dataset. The separation of training, validation and testing datasets is described in Subsection 8.5.

The main steps of the testing phase are the following:

1. Read the bidirectional flows from the Argus file.
2. Separate the 4-tuples.
3. Assign the manual ground-truth labels to each 4-tuple in the testing dataset.
4. Generate the chain of states for each 4-tuple.
5. Read the already trained behavioral models from disk. (each model is a candidate label to be applied)
6. Select the best model for each 4-tuple using a probability threshold.

7. Assign the final label to all the flows in the 4-tuple that was analyzed.
8. Output a labeled bidirectional flow file to disk.
9. Compute the error metrics.

The first three steps are the same as for the training methodology, but applied to the testing dataset. Without the assignment of ground-truth labels to the testing dataset it is not possible to compute the final error metrics. The sixth step, (selection of the best model) is a core part of our detection method because its where the matching between the unknown behaviors and the trained models is done, and is described in Subsection 6.2.4.2. The usage of the probability threshold is explained in SubSection 6.2.4.2. Once the best model was selected, the final label is known and it is assigned to all the flows in the 4-tuple.

The next Subsection describes the computation of the error metrics. The error metrics help evaluate the performance of a detection method and help understand the results.

6.2.7 Computing the Error Metrics

The methodology used by any detection method to compute its error metrics is very important and should be carefully described. It is important because the metrics are susceptible to involuntary errors. Along with the creation of the dataset, the error metrics may allow to tune the results by redefining how the errors are computed. This is the reason why it should be correctly and completely described.

The methodology of the error metrics used in our detection method is the same that we proposed before in the BClus method, and that is described in Subsection 7.4. The tool to compute the error metric is also the same, called BotnetDetectorsComparer.py [5]. In summary, the methodology proposes to compute the error metrics from the point of view of a network administrator (i.e. computing the detection of IP address and not flows), to use time frames to speed up the delivery of results and to have a correcting function that ages the metrics and therefore favors faster methods.

The methodology to compute the error metrics can be described as a series of steps. First, to separate the flows in time frames. Second, to compute the TP (True Positive), TN (True Negative), FP (False Positive) and FN (False Negative) values based on the detection of IP addresses during the time frame, and not based on the detection of flows. This means that a TP (FP) is accounted when the C&C (Normal) IP address is detected

[5] http://sourceforge.net/projects/botnetdetectorscomparer/

as C&C at least once during the time frame, and that a TN (FN) is accounted when a Normal (C&C) IP address is detected as Normal during the whole time frame. This give us new values that we called cTP, cTN, cFP and cFN.

The third step is to apply a correcting function to the values cTP, cTN, cFP and cFN in order to weight them. The correcting function is based on the idea that is better to detect a C&C IP address as a C&C (TP) earlier than latter in time, and that to miss a C&C (FN) is worst earlier that latter. The correcting function was shown in Equation 7.1.

This function depends on the ordinal number of the time frame were it is computed and on the value of α. The α value used for our comparison was 0.01 and the time frame was 300 seconds (5 minutes). After applying the correcting function we obtain the values tTP, tTN, tFP and tFN. The fourth and last step is to use these last values to compute the final error metrics, i.e. FPR, TPR, TNR, FNR, Precision, Accuracy, ErrorRate and FMeasure1.

Apart from the new way of computing the error metrics, our methodology has two main differences with other approaches. The first difference is about what we consider a True Positive. In the CCDetector method we compute a True Positive when we detect a C&C channel correctly, and not when we detect a Botnet (as in any connection of the botnet). The difference is important since, in the dataset, the labels for C&C channels are much less than the labels of all the botnets connections.

The second difference is about the number of labels in the dataset and the confusion matrix used. Since our dataset has three labels: botnet, normal and background, and since the algorithms can predict three labels: botnet, normal or background, our confusion matrix allows *nine* decisions instead of the classic *four*. Table 6.6 show how a confusion matrix does look like when the Background labels are included. The new decisions in the Table are: First, if a detection method predicts Background when the real label is Normal, then the decision should be True Negative, since the method does not output any alarm or signal preventing that the IP address should be cleaned. Second, if the detection method predicts Background when the real label is C&C, then it should be considered a False Negative, since the C&C is missed. This difference on the amount of labels and the confusion matrix should be carefully considered when creating a botnet detection method since it can help produce better results for the network administrators.

6.3 Results and Analysis

To test the performance of our method, we first had to train the behavioral models using the training dataset. As a result of the training of the CCDetector method, the

Real	Predicted Normal	Predicted Botnet	Predicted Back-ground
Normal	TN	FP	**TN**
Botnet	FN	TP	**FN**
Background	**G1**	**G2**	**G3**

TABLE 6.6: Confusion Matrix using the Normal, Botnet and Background concepts. It has nine decisions instead of four.

following ten 4-tuples were automatically selected as the behavioral models to be used for detection.

- From-Botnet-V44-TCP-CC107-IRC-Not-Encrypted
- From-Botnet-V48-TCP-CC108-Plain-HTTP
- From-Botnet-V54-TCP-CC5-Plain-HTTP-Encrypted-Data
- From-Botnet-V45-TCP-CC73-Not-Encrypted
- From-Botnet-V48-TCP-CC77-HTTP-Not-Encrypted
- From-Botnet-V54-TCP-CC6-Plain-HTTP-Encrypted-Data
- From-Botnet-V46-TCP-CC12-HTTP-Not-Encrypted
- From-Botnet-V54-TCP-CC12-HTTP-Not-Encrypted
- From-Botnet-V46-TCP-CC5-Plain-HTTP-Encrypted-Data
- From-Botnet-V54-TCP-CC16-HTTP-Not-Encrypted

Once the behavioral models were selected, the true performance of our method can be estimated by using the testing dataset. Each of the following Subsections describe the execution of our method on each of the five testing datasets.

6.3.1 Results and Analysis of Experiment 1

The experiment 1 consisted in the execution of the CCDetector method on the scenario 1 using the already trained models and a probability threshold of 0.2. The results are shown in Table 6.7. The CCDetector method was able to achieve in this experiment an FMeasure1 of 92% and a FPR of 0.5%. These results were obtained because the behavioral models present in the scenario 1 were closely matched by a good generalization of the trained models. In particular, True Positives were found for the

Scen.	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
1	87.6	254	14	0	1	0.94	0.05	0	0.86	0.96	0.03	0.92
2	52.1	97.2	2	17.7	0.74	0.98	0.02	0.25	0.96	0.88	0.11	0.84
6	0	84	0	46.7	0	1	0	1	-	0.64	0.35	0
8	0	674.8	30.5	306.2	0	0.95	0.04	1	0	0.66	0.33	-
9	173.6	254	12	274.4	0.38	0.95	0.04	0.61	0.93	0.59	0.4	0.54

TABLE 6.7: Error metrics for the CCDetector method in the testing scenarios.

label *From-Botnet-V42-TCP-CC53-HTTP-Not-Encrypted* that was detected by the model with the label *From-Botnet-V46-TCP-CC5-Plain-HTTP-Encrypted-Data*. Another TP was found for the label *From-Botnet-V42-TCP-CC6-Plain-HTTP-Encrypted-Data* (states like *11rrrrrrrrArrr*) that was detected by the model with the label *From-Botnet-V46-TCP-CC5-Plain-HTTP-Encrypted-Data* (state *41rrrrrrr*). The number of TPs shown in the Table is not directly related with the number of labels that were classified correctly because of the special error metric that we used in our method and that was described in Subsection 6.2.7.

False Positives were found on several 4-tuples of the label *From-Normal-V42-Jist* (states like *33fififf0ttccccittcccciicttcccctte*) that were misdetected by the model with the label *From-Botnet-V48-TCP-CC108-Plain-HTTP* (states like *99ciiiiizziiittcccc*) and by the model with the label *From-Botnet-V48-TCP-CC77-HTTP-Not-Encrypted*. Other FPs were found on several 4-tuples of the label *From-Normal-V42-Grill* (state *11rrrrrrrrtacccrCr*) that were misdetected by the label *From-Botnet-V46-TCP-CC5-Plain-HTTP-Encrypted-Data* (state *41rrrrrrr*). The last FP were found on the label *From-Normal-V42-Stribrek* (states like *11rrarrraaararaaaaaarrrrr*) that were misdetected by the model with the label *From-Botnet-V46-TCP-CC5-Plain-HTTP-Encrypted-Data* (state *41rrrrrrr*).

6.3.2 Results and Analysis of Experiment 2

The experiment 2 consisted in the execution of the CCDetector method on the scenario 2 using the already trained models and a probability threshold of 0.2. The results are shown in Table 6.7. The CCDetector method was able to achieve in this experiment an FMeasure1 of 84% and a FPR of 0.2%. These results were obtained because the behavioral models present in the scenario 2 were also closely matched by a good generalization of the trained models. In particular, False Positives were found for the label *From-Normal-V43-Stribrek* (state *91iafcggiggg*) that was misdetected by the model with the label *From-Botnet-V48-TCP-CC108-Plain-HTTP* (state *99ciiiiizziiittcccc*). Other FP were found for the label *From-Normal-V47-Jist* that was misdetected by the model with label *From-Botnet-V48-TCP-CC77-HTTP-Not-Encrypted*.

6.3.3 Results and Analysis of Experiment 3

The experiment 3 consisted in the execution of the CCDetector method on the scenario 6 using the already trained models and a probability threshold of 0.2%. The results are shown in Table 6.7. In this experiment, the method did not detect a single True Positive using a probability threshold of 0.2. However, the method also had 0 False Positives, which mean that the trained models were not able to detect any botnet but did not missdetect any normal host either. There was only one labeled C&C channel in this scenario, that was assigned the label *From-Botnet-V47-TCP-CC73-Not-Encrypted* and had a behavioral state model like *71rrrrrrraaarrrrrwrrrraarrrrrwarrrarrrrrrrrarrrraarrrrrrr(...)*. This experiment is an example of a network capture were our method does not have the appropriate model to detect the botnets. In [7] the authors stated that due to the concept drift problem [99], the characteristics of new bots may not be captured during training.

6.3.4 Results and Analysis of Experiment 4

The experiment 4 consisted in the execution of the CCDetector method on the scenario 8 using the already trained models and a probability threshold of 0.2. The results are shown in Table 6.7. In this experiment, as in the experiment 3, the method did not detect any True Positive, but unlike the previous experiment it did detect some False Positives. The scenario 8 has only one C&C channel with the label *From-Botnet-V49-TCP-CC75-HTTP-Custom-Port-Not-Encrypted-Non-Periodic* and the behavioral state model like *33Cccccccccccccc*CCcCttttcccctttcttccccccccccccccccccccCCcttcctcc(...)*. Our analysis of the results showed that there is no model in the training set that could match this pattern because no botnet in the training dataset behaved like that. This means that the models in the training dataset were not enough to capture this behavior, but they did misdetect some normal traffic. The final FPR was 0.4% which is still considerably low. This experiment is another example of our model not being able to detect a C&C because we didn't see the required behavior before. We believe that with the correct model in the training dataset the results can be better.

6.3.5 Results and Analysis of Experiment 5

The experiment 5 consisted in the execution of the CCDetector method on the scenario 9 using the already trained models and a probability threshold of 0.2. The results are shown in Table 6.7, where it can be seen that the method achieved a FMeasure1 of 54% with a FPR of 0.4%. This scenario is the largest of the testing dataset because it

includes ten bots and a large amount of C&C behaviors. Therefore, the experiment generated several True Positives and several False Negatives, consequently lowering the FMeasure1 metric.

6.4 Conclusions of the CCDetector Method

In this Chapter we presented a new behavioral model of network traffic and a new Markov Chain-based botnet detection method. The behavioral model is based on states and it can accurately represent the transitions between states of any network connection. The detection method, called CCDetector, uses this model for detecting similar behavioral patterns in unknown network. The CCDetector method models the behavior of the known C&C channels in the training dataset using the size of the flow, the duration of the flow and the 2nd order time difference of the flow (periodicity). Then it detects similar behavioral patterns by comparing the trained Markov Models with the unknown network traffic. The key concepts of the Chapter are the behavioral state model and the Markov Chain model for botnet detection. The real models of known behaviors that were trained are an asset that can be further extended with more botnet captures and may be used in new detection methods. The more behavioral models we can capture, the better results we can achieve.

The highlights of the CCDetector method are:

- The behavioral state model of network traffic.
- The training and storage of known botnet behaviors.
- The detection method based on Markov Chains, that can generalize.

The next Chapter compares the results obtained with the CCDetector method with other three detection methods.

Chapter 7

Comparison of the BCLus and CCDetector Methods. A Collaboration with the CTU University in Czech Republic

7.1 Introduction

One of the goals of every detection proposal is to somehow improve the results achieved so far in the area. For this purpose, new proposals present a detection method, a dataset and some results. The first problem arises when trying to show that the results are good. It is usually difficult to say if some results are good or not because it depends on the meaning of *good* for the authors. The second issue arises when trying to measure the results with some error metrics. The selection of the error metrics depend on what you want to measure and why. At the end it is equally difficult to know if the error metrics are good, just as with the results. The most common way of knowing if the results were good is by comparing them with other detection methods. That way it is possible to have a better idea of their significance.

However, the comparison of detection methods is a difficult task and needs to be done correctly [30] [100]. Common problems to solve are the use of a common dataset with a format that every method can read, a common output format for the results, an agreed meaning for the results and more important, a common error metric and comparison criteria. In summary, it should be decided what it means to be *best*. It is so hard to compare methods correctly that it may be the cause of why most botnet

detection methods are usually presented without any comparison. As far as we know only four papers made such a comparison so far [30, 49, 65, 66]. Although it is generally accepted that more comparisons with third-party methods may help to improve the area, few papers were able to do it. Among the factors that may have prevented more comparisons are the difficulties to share a botnet dataset, the lack of a good and common botnet dataset [9], the absence of a proper description of the methods [64] and the lack of a comparison methodology [100]. As stated by Rossow et al. [9], the error metrics used on most papers are usually non-homogeneous. Papers tend to use different error metrics and different definitions of error. Among the error metrics reported by the proposals in the area, the most common is the False Positive Ratio (FPR). However, this metric alone is not enough to fully compare botnet detection methods. A network administrator is usually the person in charge of applying the detection methods and dealing with the wrong results. Therefore, it could help to take their needs into account. The FPR metric alone can not measure all the needs of a network administrator. A network administrator usually needs to know the IP addresses of the infected machines without waiting too much time. This was our motivation to create a new error metric specifically designed to compare botnet detection methods.

The aim of this chapter is to compare the results of our BClus and CCDetector methods with two other third-party botnet detection methods: CAMNEP [31] and BotHunter [32]. The CAMNEP detection method is a highly sophisticated software that is being developed by the security research team of the Czech Technical University in Czech Republic and that was sold to Cisco Systems in 2013. We have been in contact with this group in the last years through a joint project between both countries. The purpose of the relationship had been to work together in a shared and common dataset for this comparison. The other method in the comparison is the BotHunter proposal. This is a well known detection method that was published in the community several years ago. It is the only publicly available detection software that uses machine learning algorithms for detection.

The comparison between the four methods was done by working with the results they produced on the same dataset. This dataset is fully described in the Subsection 8.3. The results of the four detection methods were compared using a specific methodology and a novel error metric specifically designed for the needs of the network administrators.

The contributions of this chapter are:

- A deep comparison of four detection methods. Our own algorithms BClus and CCDetector, and the third-party algorithms CAMNEP [69] and BotHunter [32].

- A generic methodology for comparing botnet detection methods along with the corresponding public tool for reproducing that methodology.
- A new error metric designed for comparing botnet detection methods.

7.2 The CAMNEP Detection Method

The first third-party method that is compared to our methods is the Cooperative Adaptive Mechanism for NEtwork Protection (CAMNEP) [69]. This system was developed by the CTU Universtiy of Czech Republic. It is a Network Behavior Analysis system [101] that consists of various state-of-the-art anomaly detection methods. The system models the normal behavior of the network and/or individual users behaviors and labels deviations from normal behaviors as anomalous.

7.2.1 Architectures of the Internal Systems of CAMNEP

CAMNEP processes the NetFlow data that is provided by routers or other network equipment. This data is analyzed to identify anomalous traffic by means of several collaborative anomaly detection algorithms. It uses a multi-algorithm and multi-stage approach to reduce the amount of false positives generated by the individual anomaly detectors without compromising the performance of the system. The self-monitoring and self-adaptation techniques, described in Subsection 7.2.4, are very important to this purpose because they help to improve the error rate of the system with a minimal and controllable impact on its efficiency.

CAMNEP consists of three main layers that evaluate the traffic: first the anomaly detectors layer; second the trust models layer; and third the anomaly aggregators layers. The anomaly detectors layer analyze the NetFlows using various anomaly detection algorithms. The system is currently using eight different anomaly detection approaches. Each of them uses a different set of features, thus looking for an anomaly from slightly different perspectives. The output of these algorithms is aggregated into events using several statistical functions and the results are sent to the trust models. All the detection algorithms used in the system are described in detail in Subsection 7.2.2.

The trust models layer maps the NetFlows into traffic clusters. These clusters group together the NetFlows that have a similar behavioral pattern. They also contain the anomaly value of the type of the event that they represent. These clusters persist over time and the anomaly value is updated by the trust model. The updated anomaly value of a cluster is used to determine the anomaly of new NetFlows. Therefore, the trust

models act as a persistent memory and reduce the amount of false positives by means of the spatial aggregation of the anomalies.

The aggregators layer creates one composite output that integrates the individual opinion of several anomaly detectors as they were provided by the trust models. The result of the aggregation is presented to the user of the system as the final anomaly score of the NetFlows. Each aggregator can use two different averaging operators: a order-weighted averaging [102] or simple weighted averaging. CAMNEP is using a simulation process to determine the best aggregation operator for the current type and state of network. This process is described in Section 7.2.4.

7.2.2 Anomaly detectors

The anomaly detectors in the CAMNEP system are based on already published anomaly detection methods. These detectors work in two stages: (*i*) they extract meaningful features associated with each NetFlow (or group of NetFlows), and (*ii*) they use the values of these features to assign an anomaly score to each NetFlow. This anomaly score is a value in the $[0, 1]$ range. A value of 1 represents an anomaly and a value of 0 represents normal behavior. The model of the anomaly detector is a fuzzy classifier that provides an anomaly value for each NetFlow. This value depends on the NetFlow itself, on the other NetFlows in the current context and on the internal traffic model, which is based on the past traffic observed on the network. The following subsections describe each of the anomaly detectors used in the CAMNEP system.

7.2.2.1 MINDS

The MINDS algorithm [103] builds a context information for each evaluated NetFlow using the following features: the number of NetFlows from the same source IP address as the evaluated NetFlow, the number of NetFlows toward the same destination host, the number of NetFlows towards the same destination host from the same source port, and the number of NetFlows from the same source host towards the same destination port. This is a simplified version of the original MINDS system, which also uses a secondary window defined by the number of connections in order to address slow attacks. The anomaly value for a NetFlow is based on its distance to the normal sample. The metric defined in this four-dimensional context space uses a logarithmic scale on each context dimension, and these marginal distances are combined into the global distance as the sum of their squares. In the CAMNEP implementation of this algorithm, the variance-adjusted difference between the floating average of past values

and the evaluated NetFlow on each of the four context dimensions is used to know if the evaluated NetFlow is anomalous. The original work is based on the combination of computationally-intensive clustering and human intervention.

7.2.2.2 Xu

In the algorithm proposed by Xu et al. [104], the context of each NetFlow to be evaluated is created with all the NetFlows coming from the same source IP address. In the CAMNEP implementation, for each context group of NetFlows, a 3 dimensional model is built with the normalized entropy of the source ports, the normalized entropy of the destination ports, and the normalized entropy of the destination IP addresses. The anomalies are determined by some classification rules that divide the traffic into normal and anomalous. The distance between the contexts of two NetFlows is computed as the difference between the three normalized entropies, combined as the sum of their squares. The implementation of the algorithm used in CAMNEP is close to the original publication, which was further expanded by Xu and Zhang [105], except for the introduction of new rules that define an horizontal port scan as anomalous.

7.2.2.3 Lakhina Volume

The volume prediction algorithm presented in Lakhina et al. [106] uses the Principal Components Analysis (PCA) algorithm to build a model of traffic volumes from individual sources. The observed traffic for each source IP address with non-negligible volumes of traffic is defined as a three dimensional vector: the number of NetFlows, number of bytes and number of packets from the source IP address. The traffic model is defined as a dynamic and data-defined transformation matrix that is applied to the current traffic vector. The transformation splits the traffic into normal (i.e. modeled) and residual (i.e. anomalous). The transformation returns the residual amount of NetFlows, packets and bytes for each source IP address. These values define the context (identical for all the flows from the given source). An anomaly is determined by transforming the 3D context into a single value in the $[0, 1]$ interval.

Notice that the original work was designed to handle a different problem, that is, the detection of anomalies on a backbone network. Also, the original work modeled networks instead of source IP addresses. However, the implementation in CAMNEP is a modified version that obtains a classifier that can successfully contribute to the joint opinion when combined with others.

7.2.2.4 Lakhina Entropy

The entropy prediction algorithm presented by Lakhina et al. [107] is based on the similar PCA-based traffic model than Subsection 7.2.2.3, but it uses different features. It aggregates the traffic from the individual source IP addresses, but instead of traffic volumes it predicts the entropies of destination IP addresses, destination ports and source ports over the set of contextual NetFlows of each IP source address. The context space is therefore three dimensional. An anomaly is determined as the normalized sum of the residual entropy over the three dimensions. The metric is simple: a function measures the difference of residual entropies between the NetFlows and aggregates their squares. Also, the original anomaly detection method was significantly modified along the same lines as the volume prediction algorithm.

7.2.2.5 TAPS

The TAPS method [108] is different from the previous approaches because it targets horizontal and vertical port scans. The algorithm only considers the traffic sources that created at least one single-packet NetFlow during a particular observation period. These preselected sources are then classified using the following three features: number of destination IP addresses, number of destination ports and the entropy of the NetFlow size measured in number of packets. The anomaly value of the source IP address is based on the ratio between the number of unique destination IP addresses and destination ports. When this ratio exceeds a predetermined threshold the source IP address is considered as a scan origin. Using the original method there were an unusually high number of false positives. Therefore, the implementation in CAMNEP extends the original method with the entropy of the NetFlow size to achieve better results.

7.2.2.6 KGB

The KGB anomaly detector presented by Pevny et al. [109] is also based on Lakhina's work. It uses the same features as Lakhina Entropy detector described above. Similar to Lakhina's work, it performs a PCA analysis of the feature vectors for each source IP address in the dataset. The final anomaly is determined from the deviations of averaging the principal components. There are two versions of KGB detector. The KGBf version that examines principal components with high variances and the KGBfog version that examines principal components with low variances.

7.2.2.7 Flags

The Flags detector uses the same detection method as the KGB detector [109]. The only difference is on the feature input vector. The feature vector of the Flags detector is determined by the histogram of the TCP flags of all the NetFlows with the same IP source address. This detector is looking for a sequence or a combination of anomalous TCP flags.

7.2.3 Trust Modeling

The trust models are specialized knowledge structures studied in multi-agent research [110, 111]. The features of trust models include fast learning, robustness in response to false reputation information and robustness with respect to environmental noise. Recent trust models, inspired by machine learning methods [112] and pattern recognition approaches [113] make the trust reasoning more relevant for network security, as they are able to:

- include the context of the trusting situation into the reasoning, making the trust model situational;
- use the similarities between trustees to reason about short-lived or one shot trustees, such as NetFlows.

In this context, a *feature vector* includes the identity of a NetFlow and the context of the NetFlow by each trust model in the feature space. The term *centroid* is used to denote the permanent feature vectors that are positioned in the feature spaces of trust models. The centroids act as trustees of the model, and the trustfulness value of each centroid is updated. Each centroid is used to deduce the trustfulness of the feature vectors in its vicinity.

The anomaly detectors integrate the anomaly values of individual NetFlows into their trust models. The reasoning about the trustfulness of each individual NetFlow is both computationally unfeasible and unpractical (the NetFlows are single shot events by definition), and thus the centroids of the clusters holds the trustfulness of significant NetFlow samples. The anomaly value of each NetFlow is used to update the trustfulness of centroids in its vicinity. The weight used for the update of the trustfulness of the centroids decreases with the distance. Therefore, as each model uses a distinct distance function, they all have a different insight into the problem.

Each trust model determines the *trustfulness* of each NetFlow by finding all the centroids in the NetFlows vicinity. It sets the trustfulness using the distance-based weighted average of the values preserved by the centroids. All the models provide their trustfulness assessment (conceptually a reputation opinion) to the anomaly aggregators.

7.2.4 Adaptation

The adaptation layer of CAMNEP identifies the optimal trustfulness aggregation function that achieves the best separation between the legitimate and malicious NetFlows. This layer is based on the insertion of challenges into the NetFlow data observed by the system. The challenges are NetFlows of past classified incidents. They are generated by short lived, challenge specific *challenge agents* and are mixed with the input traffic. They cannot be distinguished from the rest of the input traffic by the detectors/aggregators. They are processed and evaluated with the rest of the traffic. Also, they are used to update the anomaly detection mechanisms and the trust models. Once the process is completed, the challenges are re-identified by their respective challenge agents and removed from the output. The anomaly value given to these NetFlows by the individual anomaly aggregators is used to evaluate those aggregations and to select the optimal output for the current network conditions.

There are two broad types of challenges: The *malicious challenges* that correspond to known attacks, whereas the *legitimate challenges* represent known instances of legitimate events that tend to be misclassified as anomalous. Malicious challenges are further divided into broad attack classes, such as fingerprinting/vertical scan, horizontal scan, password brute forcing, etc. For each attack class, each aggregator has a probability distribution that is empirically estimated from the continuous anomaly values attributed to the challenges in that class. All the legitimate challenges are also defined by a distribution.

The CAMNEP system assumes that the anomaly values of both the legitimate and malicious challenges are defined by normal distributions[1]. The distance between the estimated mean normalized values of both distributions represents the quality of the aggregator with respect to a given attack class. The *effectiveness* of the aggregator, defined as an ability to distinguish between the legitimate events and the attacks is defined as a weighted average of the effectiveness with respect to individual classes.

As the network traffic is highly dynamic, it is very difficult to predict which aggregation function will be chosen, especially given the fact that the challenges are selected from

[1]The normality of both distributions is not difficult to achieve provided that the attack classes are properly defined and that the challenge samples in these classes are well selected, i.e. comparable in terms of size and other parameters.

a challenge database using a stochastic process with a pseudo-random generator unknown to a potential attacker. The attacker therefore faces a dynamic detection system that unpredictably switches its detection profiles. Each profile has a utility value (i.e. detection performance) close to the optimum. This unpredictability, together with the additional robustness achieved by the use of multiple algorithms, makes the evasion attempt a much more difficult task than simply avoiding a single intrusion detection method [114].

The system is able to find the optimal thresholds for the anomaly score when using the results from the adaptation process. It is continuously modeling the normal distribution of malicious and legitimate challenges. The threshold is set to minimize the Bayes risk (posteriori expected loss) computed from the modeled legitimate and malicious behavior distributions. Thus, the final result of the system is a list of NetFlows with the label anomalous or normal.

7.2.5 Training of the CAMNEP Method

Since the system needs the inner models of the anomaly detectors and trust models to have the optimal detection results, it is necessary to train them. Typically, the system needs 25 minutes of traffic to create its inner models and to adapt itself to the current type of network and its state. Therefore, the training data for the CAMNEP algorithm was created by trimming off some minutes at the start of each of the scenarios in the dataset described in Section 8.3.

7.3 The BotHunter Detection Method

The second third-party method that is compared to our methods is the BotHunter method. This method was proposed by Gu et al. [32] to detect the infection and coordination dialogue of botnets by matching a state-based infection sequence model. It consists of a correlation engine that aims at detecting specific stages of the malware infection process, such as inbound scanning, exploit usage, egg downloading, outbound bot coordination dialog and outbound attack propagation.

BotHunter uses an adapted version of the Snort IDS[2] with two proprietary plug-ins, called *Statistical Scan Anomaly Detection Engine* (SCADE) and *Statistical Payload Anomaly Detection Engine* (SLADE). SLADE implements a lossy n-gram payload analysis

[2] http://www.snort.org

of incoming traffic flows to detect divergences in some protocols. SCADE performs port scan analyzes.

An infection is reported when one of two conditions is satisfied: first, when an evidence of local host infection is found and evidence of outward bot coordination or attack propagation is found, and second, when at least two distinct signs of outward bot coordination or attack propagation are found. The BotHunter warnings are tracked over a temporal window and contribute to the infection score of each host.

The BotHunter proposal is compared to the BClus, CCDetector and CAMNEP methods to have the reference of an accepted detection method in the community. The version of BotHunter used in the comparison is 1.7.2.

7.4 Comparison Methodology and New Error Metric

In order to be repeatable and extendable, the comparison of detection methods should have a methodology. For this purpose we created a simple methodology and a new error metric. The methodology may be used by other researchers to add the results of their methods to our comparison. The new error metric is used to adapt the significance of the metrics to the needs of the network administrator. The Subsection 7.4.1 presents the comparison methodology and ths Subsection 7.4.2 presents the error metric.

7.4.1 Comparison Methodology

The idea of comparing detection methods to verify their performance is good and useful. When researchers develop a method, they usually want to compare it to other third-party methods. However, this comparison can prove very difficult to accomplish. The first limitation is that a complete description of the third-party method is not usually available. The authors of the third-party method may have no space in the paper to explain all the details, or maybe the method is so complex that it is really difficult to explain it. The smallest information missing from the description, such as the values of the thresholds, is enough to forbid a comparison. The second limitation is that the dataset used by the third-party methods is usually not available. The most common reason is that the authors do not have the authorization to publish it because of privacy issues. Finally it may happen that the method is not fully published because the authors decided to commercialize it. It can be safely concluded that having an original third-party method and its dataset for comparison purposes is very difficult.

Even if the third-party method and dataset can be obtained, it is not straightforward to actually make the comparison. The methods to be compared need to read the same type of input and generate the same type of output in order to compute the same error metrics. The comparison methodology that we used in this chapter implied to sent the common dataset to the third-party research team, so they can run the CAMNEP method on it. In this way we didn't need to have access to the implementation details of CAMNEP or their internal dataset.

The following is a summary of the comparison methodology that we used:

- Agree with the third-party researchers upon a common input and output.
- Each group has access to the same common dataset, formatted as the agreed input.
- Each group run its method on the dataset. Avoiding any privacy issue on the details of the methods.
- The methods output the same dataset with an extra column for the prediction.
- The datasets with the predictions are shared between the groups.
- Each group can compute the error metrics by its own.

The idea of a common output format is that each method only adds a column with its label predictions to the original dataset. Therefore, the dataset can be analyzed by automatic tools to compute the error metrics. Since our dataset is composed on NetFlows, it is easy to add a new column to each NetFlow line. In this way, more and more methods can publish their results on the same dataset and more comparisons can be made. The main advantage of this approach is that the details of the methods remain private. The main disadvantage is that the research teams must agree to cooperate.

To implement this methodology and automatically compute the error metrics for all the methods we created and published a new tool called *Botnet Detectors Comparer* (Garcia et al. [30]). This tool is publicly available for download[3]. The error metric computed by this tool is described in next Subsection 7.4.2.

The Botnet Detectors Comparer tool reads the common dataset with a NetFlow format and implements the following steps:

- Separates the NetFlow file in *comparison time windows*.

[3] http://downloads.sourceforge.net/project/botnetdetectorscomparer/BotnetDetectorsComparer-0.9.tgz

- Compares the ground-truth NetFlow labels with the predicted labels of each method and computes the TP, TN, FP and FN values.
- After the *comparison time window* ended, it computes the error metrics: FPR, TPR, TNR, FNR, Precision, Accuracy, ErrorRate and FMeasure1 for that time window.
- When the dataset ends, it computes the final error metrics.
- The error metrics are stored in a text file and plotted in a eps image.

The *comparison time window* is the time window used for computing the error metrics and it is not related with the methods. It is the time that the network administrator may wait to have a decision about the traffic. In our methodology the width of the *comparison time windows* was five minutes.

Using this methodology, researchers can now add its own predictions to the NetFlows files of our dataset and use this tool to compute the error metrics. The next Subsection describes the new error metric proposed to compare botnet detection methods.

7.4.2 Our New Error Metric

The usual steps to verify a detection method are to get some results, to obtain the errors on the results, i.e. False Positives, False Negatives, True Positives and True Negatives, and then obtain the error metrics, such as FPR (False Positive Rate), FNR (False Negative Rate), TPR (True Positive Rate), etc. These error metrics were historically designed from a statistical point of view, and they are really good to measure differences and to compare most methods. However, the needs of a network administrator that is going to use a detection method in a real network are slightly different from the needs of a researcher. The issue with the classic error metrics is that they do not have a meaning that can be easily translated to the network [89]. This has been called the semantic gap [9]. For example, and according to the classic definition, a False Positive should be detected every time that a normal NetFlow is detected as *botnet*. However, a network administrator might want to detect a small amount of infected IP addresses instead of hundreds of NetFlows. Furthermore, she may need to detect them as soon as possible. We thought that there was a need for a new error metric to compare detection methods from the point of view of network administrators.

Based on this idea and together with Martin Grill from CTU University, we created a new set of error metrics that adhere to the following principles:

- Errors should account for IP addresses instead of NetFlows.

- To detect a botnet IP address (TP) early is better than latter.
- To miss a botnet IP address (FN) early is worst than latter.
- The value of detecting a normal IP address (TN) is not affected by time.
- The value of missing a normal IP address (FP) is not affected by time.

The first characteristic of our error metric is to incorporate the *time* to the errors by computing them in *comparison time frames*. These time frames are only used to compute the errors and are independent of the detection methods. This means that instead of computing the FPR along a four month-long capture, we compute the FPR every some minutes.

The second characteristic of our error metric is a migration from a NetFlow-based detection to an IP-based detection. The classical error values (TP, FP, TN, FN) were redefined as follows:

- c_TP: A True Positive is accounted when a Botnet IP address is detected as Botnet at least **once** during the comparison time frame.
- c_TN: A True Negative is accounted when a Normal IP address is detected as Non-Botnet during the **whole** comparison time frame.
- c_FP: A False Positive is accounted when a Normal IP address is detected as Botnet at least **once** during the comparison time frame.
- c_FN: A False Negative is accounted when a Botnet IP address is detected as Non-Botnet during the **whole** comparison time frame.

The third characteristic is a modification in the values of the error metrics by adding a time-based *correcting function* that is defined as follows:

$$correcting\_function = e^{(-\alpha * N^{\circ}\ comparison\ time\ frame)} + 1 \tag{7.1}$$

This correcting function depends on the ordinal number of the comparison time frame were it is computed and on the value of α. Figure 7.4.2 shows this monotonically decreasing function that weights the values according to the comparison time frame number. The main idea is to weight the values more on the firsts time frames and to weight them less on the lasts ones. The α value is used to manually fit the function according to the capture length. The α value used for our comparison was 0.01.

Using this $correcting\_function$, four time-dependent new error metrics were created, called tTP, tTN, tFP and tFN. Note that they use the previously defined IP-based error metrics. They are computed as follows:

- tTP:

$$\frac{c\_TP * correcting\_function}{N^{\circ} \text{ of unique botnet IP addresses in the comparison time frame}} \tag{7.2}$$

- tFN:

$$\frac{c\_FN * correcting\_function}{N^{\circ} \text{ of unique botnet IP addresses in the comparison time frame}} \tag{7.3}$$

- tFP:

$$\frac{c\_FP}{N^{\circ} \text{ of unique normal IP addresses in the comparison time frame}} \tag{7.4}$$

- tTN:

$$\frac{c\_TN}{N^{\circ} \text{ of unique normal IP addresses in the comparison time frame}} \tag{7.5}$$

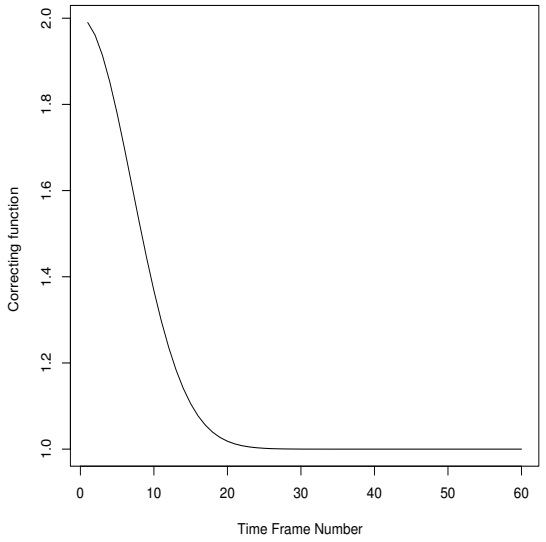

FIGURE 7.1: Correcting Function applied to 60 time windows.

These time-based error allow for a more realistic comparison between detection algorithms. An algorithm weights better if it can detect sooner all the infected IP addresses without error. To miss an infected IP address at the beginning is more costly than to falsely detect an infected IP address at the beginning. After some time frames, all the error values are weighted the same.

With these time-based errors we can now compute the new corresponding error rates. They are like the classic ones, but are redefined to use the time-based errors:

- $FPR = \frac{tFP}{tTN+tFP}$
- $TPR = \frac{tTP}{tTP+tFN}$
- $TNR = \frac{tTN}{tTN+tFP}$
- $FNR = \frac{tFN}{tTP+tFN}$
- $Precision = \frac{tTP}{tTP+tFP}$
- $Accuracy = \frac{tTP+tTN}{tTP+tTN+tFP+tFN}$
- $ErrorRate = \frac{tFN+tFP}{tTP+tTN+tFP+tFN}$
- $FMeasure1 = 2 * \frac{Precision*TPR}{Precision+TPR}$ (FMeasure with beta=1)

These error metrics are implemented in the *Botnet Detectors Comparer* tool that it is publicly available for download and described in the previous Subsection.

7.5 Comparison of the Results of the Detection Methods

To compare the four detection methods described in this Chapter, each of them was executed on the five testing Scenarios described in Subsection 8.3. Each method added a new column to each Scenario with the predicted labels for each NetFlow. The separation of the Scenarios in training and testing is explained in Subsections 8.3 and 8.5. The Bclus, BotHunter and CAMNEP methods were executed using exactly the same NetFlows files. The CCDetector method, however, was executed on NetFlow files that had two differences: First, that the flows were bidirectional instead of unidirectional. Second, that the dataset included more detailed ground-truth labels (but the same amount of labeled flows). Technically all the methods used the same dataset, but the CCDetector method used a different representation.

To better understand the implications of comparing these methods, the following baseline algorithms were added: the *AllPositive* algorithm, that always predicts Botnet, the *AllNegative* algorithm that always predicts Normal and the *AllBackground* algorithm that always predicts Background. They are analyzed alongside with the BClus, CCDetector, CAMNEP and BotHunter algorithm. The names of all the algorithms compared are encoded in Table 7.1. Apart from the four main compared algorithms and the baseline algorithms, this Table shows all the internal algorithms used by CAMNEP.

The comparison of the results was done by reading the ground-truth label of each NetFlow and comparing it to the predicted label for that NetFlow by each method. However, the BotHunter method does not read NetFlows files and does not output a prediction label for each NetFlow, making the comparison more difficult. To solve this issue we run BotHunter on the original pcap files and we obtained, for each pcap, a list of alerts. These alerts include the date, the name of the alert, the protocol, the source IP address, source port, the destination IP address and destination port. Then, we searched which was the NetFlow corresponding to that alert and we assigned it the label *Botnet*. The rest of the NetFlows were labeled as *Normal*. With this label assignment procedure it was possible to add the BotHunter method to the comparison. The next Subsections compare the results on each of the five testing scenarios of the dataset.

7.5.1 Comparison of Results in Scenario 1

This scenario corresponds to an IRC-based botnet that sent spam for almost six and a half hours. The error metrics for this comparison are shown in Table 7.2, which is ordered by FMeasure1. The Table shows that the AllPositive algorithm had an FMeasure1 of 65%, although it had a 100% FPR. The BClus algorithm had a FMeasure1 of 48% and a FPR of 40%. The BotHunter algorithm had a FMeasure1 of 2% and a FNR of 98%. The CAMNEP (CA1) algorithm had a very low FMeasure1 and very low FPR. In this Table it can be seen that the CCDetector method achieved a really good FMeasure1 compared with the rest, even with our old BClus method. Also, the False Positive Ratio achieved by CCDetector is the smallest. Moreover, the TPR of CCDetector method is 1, which means that it did not miss any botnet at all. We consider that the results achieved by CCDetector in this experiment are the best and they support the idea that behavioral models can be used to better detect botnets.

A simplified comparison between the BClus, CCDetecor, CAMNEP and BotHunter algorithms is shown in Figure 7.2. The BClus algorithm had between 40% and 60% for the TPR, FPR, TNR and FNR metrics and nearly 50% for the FMeasure1. The CAMNEP algorithm had a value near 0% for the TPR, near 1% for the TNR and near 0% for the FMeasure1. The CCDetector method had the best combination of values, obtaining a TPR of 92%.

The apparently good results of the AllPositive algorithm may have an explanation. This algorithm predicts always Botnet, which gives a TPR of 100%, a precision of 50% and a FMeasure1 of 66%. The only traffic that this algorithm can mis-classify is the Normal traffic. However, the amount of Normal traffic in this scenario is considerably smaller than the rest of the labels. This imbalance made the AllPositive algorithms have better

Algorithm Name	Reference
Flags-FOG-srcIP.src.fog-1.00	Fs1
Flags-FOG-srcIP.src.fog-1.50	Fs1.5
Flags-FOG-dstIP.dst.fog-1.00	Fd1
Flags-FOG-srcIP.src.fog-2.00	Fs2
Flags-FOG-dstIP.dst.fog-1.50	Fd1.5
Flags-FOG-dstIP.dst.fog-2.00	Fd2
Minds-1.00	Mi1
Xu-1.00	X1
Xu-1.50	X1.5
Minds-1.50	Mi1.5
Minds-2.00	Mi2
LakhinaEntropyGS-1.00	Le1
KGBFog-sIP.src.fog-1.00	Ko1
KGBFog-sIP.src.fog-1.50	Ko1.5
MasterAggregator-1.00	**CA1**
TAPS3D-1.50	T1.5
TAPS3D-1.00	T1
KGBF-sIP.src.f-1.00	K1
KGBF-sIP.src.f-2.00	K2
AllNegative	AllNeg
AllPositive	AllPo
BClus	**BClus**
XuDstIP-1.50	Xd1.5
XuDstIP-2.00	Xd2
TAPS3D-2.00	T2
LakhinaVolumeGS-1.50	Lv1.5
MasterAggregator-1.50	CA1.5
LakhinaVolumeGS-2.00	Lv2
MasterAggregator-2.00	CA2
Xu-2.00	X2
KGBF-sIP.src.f-1.50	K1.5
XuDstIP-1.00	Xd1
LakhinaEntropyGS-1.50	Le1.5
LakhinaEntropyGS-2.00	Le2
LakhinaVolumeGS-1.00	Lv1
KGBFog-sIP.src.fog-2.00	Ko2
AllBackground	AllBac
BotHunter	**BH**
CCDetector	**CCD**

TABLE 7.1: Name reference of the algorithms used in the comparison.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
CCD	**87.6**	**254**	**14**	**0**	**1**	**0.94**	**0.05**	**0**	**0.86**	**0.96**	**0.03**	**0.92**
AllPo	65.5	0	69	0	1	0	1	0	0.4	0.4	0.5	0.65
BClus	**30.2**	**41.3**	**27.6**	**35.3**	**0.4**	**0.5**	**0.4**	**0.5**	**0.5**	**0.5**	**0.4**	**0.48**
Fs1	7.8	66.4	2.5	57.5	0.1	0.9	<	0.8	0.7	0.5	0.4	0.20
Fs1.5	6.3	67.2	1.7	59.1	<	0.9	<	0.9	0.7	0.5	0.4	0.17
Fd1	6.8	54.2	14.6	58.6	0.1	0.7	0.2	0.8	0.3	0.4	0.5	0.15
Fs2	4	67.6	1.3	61.4	<	0.9	<	0.9	0.7	0.5	0.4	0.11
Fd1.5	4.6	57.5	11.4	60.8	<	0.8	0.1	0.9	0.2	0.4	0.5	0.11
Fd2	2.2	59.8	9.1	63.2	<	0.8	0.1	0.9	0.1	0.4	0.5	0.05
Mi1	2.3	52.3	16.6	63.1	<	0.7	0.2	0.9	0.	0.4	0.5	0.05
X1	1.7	68.6	0.3	63.6	<	0.9	<	0.9	0.8	0.5	0.4	0.05
X1.5	1.5	68.6	0.3	63.9	<	0.9	<	0.9	0.8	0.5	0.4	0.04
BH	**1.59**	**73.8**	**0.18**	**109**	**0.01**	**0.9**	**<**	**0.9**	**0.8**	**0.4**	**0.5**	**0.02**
Mi1.5	1	56.9	12	64.4	<	0.8	0.1	0.9	<	0.4	0.5	0.02
Mi2	0.6	63.1	5.8	64.8	<	0.9	<	0.9	<	0.4	0.5	0.01
Le1	0.2	68.1	0.8	65.2	<	0.9	0.01	0.9	0.2	0.5	0.4	0.007
Ko1	0.1	68.7	0.1	65.3	<	0.9	<	0.9	0.4	0.5	0.4	0.004
Ko1.5	0.08	68.9	0.02	65.3	<	1	0	0.9	0.7	0.5	0.4	0.002
CA1	**0.005**	**68.7**	**0.2**	**65.4**	**0**	**0.9**	**<**	**1**	**<**	**0.5**	**0.4**	**<**
T1.5	0.005	68.9	0	65.4	0	1	0	1	1	0.5	0.4	<
T1	0.005	68.9	0	65.4	0	1	0	1	1	0.5	0.4	<

TABLE 7.2: Comparison of the error metrics in Scenario 1. FM1 is the FMeasure1. The < symbol means that the value is lower than 0.001.

results than it should. This algorithm is useful as a baseline for evaluating detection methods and datasets, but it is useless in a real network.

To better appreciate the inner workings of the detection methods during the analysis of this scenario, we plotted the accumulated and running error metrics for each comparison time frame in Figure 7.3. The running error metrics are the metrics that are computed on each time window, opposite to the metrics that are only computed at the end of the experiment. The CCDetector method is not included in this plot because it was executed in a slightly different bidirectional dataset. Figure 7.3 shows that the FPR of the BClus method was high on the first time frames, but after that it kept going down until the final 40%. On the sixth time frame the BClus method started to detect botnets with a 100% TPR until near the twelfth time frame. While still having a huge amount of FP, the BClus method managed to have a final FMeasure1 of 48%. The CAMNEP and BotHunter algorithms had low values during all the scenario.

7.5.2 Comparison of Results in Scenario 2

In this scenario, the same IRC-based botnet as scenario 1 sent SPAM for 4.21 hours. The error metrics are shown in Table 7.3. The AllPositive algorithm had an FMeasure1 of

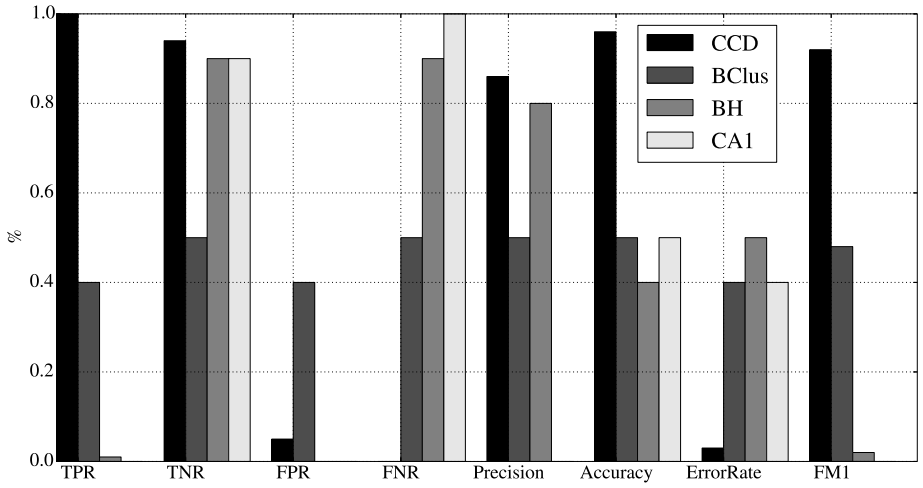

FIGURE 7.2: Simplified comparison of the error metrics for the BClus, CCDetector, CAMNEP and BotHunter algorithms on Scenario 1. FM1 is the FMeasure1.

68%. The BClus algorithm had a FMeasure1 of 41% and a FPR of 20%. The BotHunter algorithm had an FMeasure1 of 4% and a FNR of 97%. The CAMNEP algorithm had a FMeasure1 of 1% and a very small FPR. It can be seen that the CCDetector method had the best FMeasure1, with a value of 84%. As in the previous comparison of experiment one, the FPR of the CCDetector method is considerably small, meaning that, in average, only 2 computers from 100 were misdetected. Some botnet traffic was not detected (as shown by the TPR of 74%) but since there was only one botnet IP address in this scenario and the detection is being done every 5 minutes, we can be sure that the botnet IP addresses was detected. In this experiment, the CCDetector method showed that it is possible to have a good detection rate with a low error rate if we captured the correct behavioral models.

The simplified comparison for this scenario is shown in Figure 7.4. Although it was the same bot that scenario 1 and it did almost the same actions, all the algorithms gave different results. The CAMNEP method still have a large amount of tFN, but despite the low 1% FMeasure1, it was 55 times better than itself on scenario 1. Its Precision was high because there were almost no tFP, independently of the amount of tTP. Regarding the BClus method, it had a lower TPR than on scenario 1, but also a lower FPR, which lead to a comparatively better FMeasure1 value. This scenario is a good example of the variability in the network due to the presence of Background traffic. The same bot, generating the same type and amount of traffic obtained different error metrics.

The inner workings of the algorithms can be seen in the running metrics shown in Figure 7.5. The CCDetector method is not included in this plot because it was executed in a slightly different bidirectional dataset. The plot shows that the BClus method started

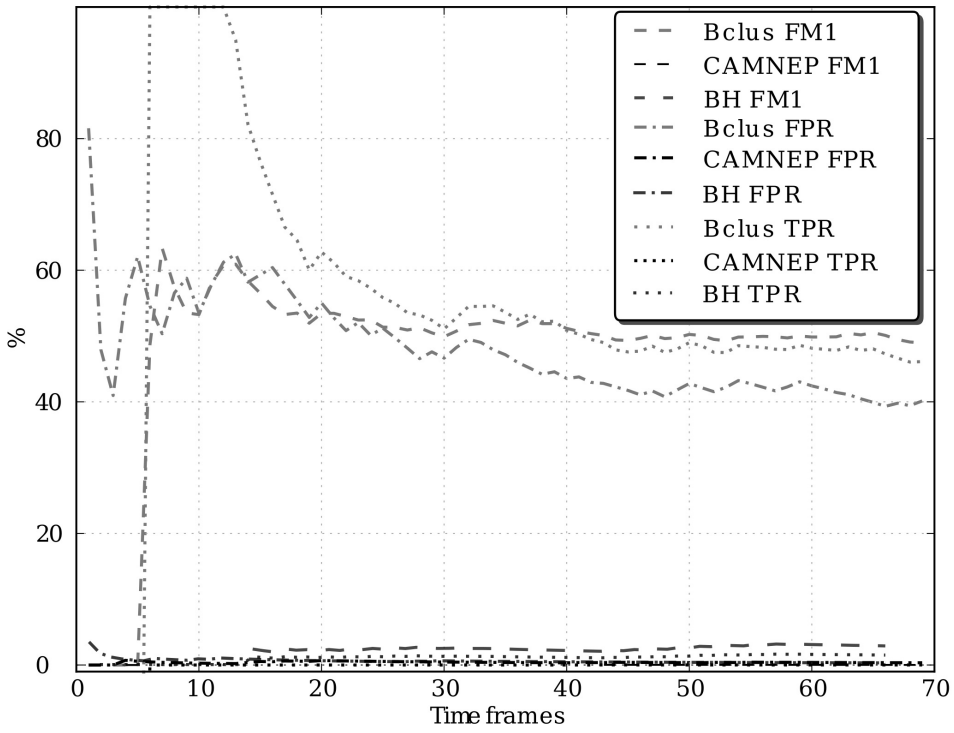

FIGURE 7.3: Comparison of the running error metrics for Scenario 1. FM1 is the FMeasure1.

with a large FPR, but after the fifth time frame it started to detect botnets correctly and its FMeasure1 value improved. The TPR, FPR and FMeasure1 values for the BClus method decreased until the end of the scenario, suggesting that the final values could be even lower. The BotHunter algorithm had a FMeasure1 close to 20% on the first time frames but then it quickly dropped to 4%. The CAMNEP error metrics remained low during the whole scenario.

7.5.3 Comparison of Results in Scenario 6

The botnet in this scenario scanned SMTP (Simple Mail Transfer Protocol) servers for two hours and connected to several RDP (Remote Desktop Protocol) services. However, it did not send any SPAM and did not attack. The C&C server used a proprietary protocol that connected every 33 seconds and sent an average of 5,500 bytes on each connection.

The error metrics for this scenario can be seen in Table 7.4. The AllPositive algorithm had a FMeasure1 of 73%, far better than the FMeasure1 of BClus and CAMNEP which were both 4%, and CCDetector which was 0%. The BotHunter algorithm had a better FMeasure1 than the BClus, CCDetector and CAMNEP algorithms because some of the

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
CCD	**52.1**	**97.2**	**2**	**17.7**	**0.74**	**0.98**	**0.02**	**0.25**	**0.96**	**0.88**	**0.11**	**0.84**
AllPo	49.9	0	47	0	1	0	1	0	0.5	0.5	0.4	0.68
BClus	**15.6**	**37.1**	**9.8**	**34.2**	**0.3**	**0.7**	**0.2**	**0.6**	**0.6**	**0.5**	**0.4**	**0.41**
Fd1	14.4	36.5	10.4	35.5	0.2	0.7	0.	0.7	0.5	0.5	0.4	0.38
Fd1.5	9.3	39.1	7.8	40.5	0.1	0.8	0.1	0.8	0.5	0.5	0.5	0.27
Fd2	7.9	40.7	6.2	42	0.1	0.8	0.1	0.8	0.5	0.5	0.4	0.24
Fs1	6.8	45.9	1	43	0.1	0.9	<	0.8	0.8	0.5	0.4	0.23
Fs1.5	6	46.3	0.6	43.8	0.1	0.9	<	0.8	0.8	0.5	0.4	0.21
X1	5.3	46.7	0.2	44.5	0.1	0.9	<	0.8	0.9	0.5	0.4	0.19
X1.5	4.3	46.8	0.1	45.6	<	0.9	<	0.9	0.9	0.5	0.4	0.15
Fs2	4.2	46.5	0.4	45.7	<	0.9	<	0.9	0.9	0.5	0.4	0.15
BH	**1.65**	**46.9**	**0.05**	**75**	**0.02**	**0.99**	**<**	**0.9**	**0.9**	**0.3**	**0.6**	**0.04**
Mi1	1.1	35.8	11.1	48.8	<	0.7	0.2	0.9	<	0.3	0.6	0.03
Mi1.5	0.6	39.1	7.8	49.2	<	0.8	0.1	0.9	<	0.4	0.5	0.02
X2	0.5	46.9	0.02	49.4	<	1	0	0.9	0.9	0.4	0.5	0.02
CA1	**0.2**	**46.9**	**0.04**	**49.6**	**<**	**0.9**	**<**	**0.9**	**0.8**	**0.4**	**0.5**	**0.01**
Ko1	0.1	46.9	0.08	49.8	<	0.9	<	0.9	0.6	0.4	0.5	0.005
Mi2	0.1	46.3	0.6	49.8	<	0.9	<	0.9	0.1	0.4	0.5	0.005
Le1	0.1	46.1	0.8	49.8	<	0.9	<	0.9	0.1	0.4	0.5	0.005
Lv1	0.1	46.6	0.3	49.8	<	0.9	<	0.9	0.2	0.4	0.5	0.004
Xd1	0.07	44	2.	49.8	<	0.9	<	0.9	<	0.4	0.5	0.003
T1	0.03	47	0	49.9	<	1	0	0.9	1	0.4	0.5	0.001

TABLE 7.3: Comparison of the error metrics in Scenario 2. FM1 is the FMeasure1. The < symbol means that the value is lower than 0.001.

IP addresses used in the SMTP connections were blacklisted as part of the RBN (Russian Business Network). It should be noted that our five testing scenarios were captured on August 2011 and the BotHunter rules were updated on January 2013, so it is possible that these IP addresses were blacklisted after the capture. The Table shows that the CCDetector method had a FMeasure1 of 0 and a FPR of 0. However, the rest of the methods also showed a very low FMeasure1 value. This means that for most of the methods it was really hard to deal with this experiment. The analysis of this scenario in Appendix A.1 show that it only had one non-periodic C&C channel. Unfortunately, none of the training models in the CCDetector could generalize enough to detect this C&C channel.

The simplified comparison for this scenario is shown in Figure 7.6. The CAMNEP and the BClus methods behaved almost identically. The only difference is that the BClus method had ten times more FPR and therefore the CAMNEP method had a better Precision and a slightly better FMeasure.

The inner workings of the algorithms can be seen in the running metrics shown in Figure 7.7. The CCDetector method is not included in this plot because it was executed in a slightly different bidirectional dataset. This Figure shows that both BClus and

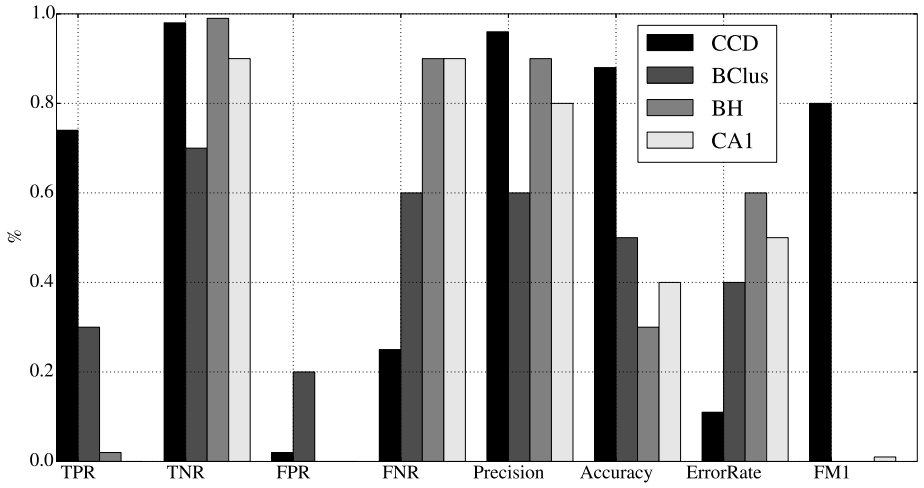

FIGURE 7.4: Simplified comparison of the error metrics for the BClus, CCDetector, CAMNEP and BotHunter algorithms on Scenario 2. FM1 is the FMeasure1.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
Fs1	22.5	20.7	0.2	6.8	0.7	0.9	<	0.2	0.9	0.8	0.1	0.86
Fd1	23.2	17.6	3.3	6	0.7	0.8	0.1	0.2	0.8	0.8	0.1	0.83
Fs1.5	20.8	20.8	0.1	8.5	0.7	0.9	<	0.2	0.9	0.8	0.1	0.82
Fd1.5	19.9	18.6	2.3	9.3	0.6	0.8	0.1	0.3	0.8	0.7	0.2	0.77
Fd2	19.1	19.3	1.6	10.1	0.6	0.9	<	0.3	0.9	0.7	0.2	0.76
Fs2	17.7	20.8	0.1	11.5	0.6	0.9	<	0.3	0.9	0.7	0.2	0.75
AllPo	29.3	0	21	0	1	0	1	0	0.5	0.5	0.4	0.73
X1	5.7	20.9	0.04	23.6	0.1	0.9	<	0.8	0.9	0.5	0.4	0.32
Xd1.5	3.6	17.3	3.6	25.7	0.1	0.8	0.1	0.8	0.5	0.4	0.5	0.19
BH	**2.53**	**20.9**	**0.02**	**37.3**	**0.06**	**0.99**	**<**	**0.93**	**0.98**	**0.38**	**0.61**	**0.11**
Xd2	3.6	17.3	3.6	25.7	0.1	0.8	0.1	0.8	0.5	0.4	0.5	0.19
Xd1	3.6	17.3	3.6	25.7	0.1	0.8	0.1	0.8	0.4	0.4	0.5	0.19
X1.5	1.4	21	0	27.8	<	1	0	0.9	1	0.4	0.5	0.09
CA1	**0.6**	**20.9**	**0.07**	**28.6**	**<**	**0.9**	**<**	**0.9**	**0.9**	**0.4**	**0.5**	**0.04**
BClus	**0.6**	**20.2**	**0.7**	**28.6**	**<**	**0.9**	**<**	**0.9**	**0.4**	**0.4**	**0.5**	**0.04**
CCD	**0**	**84**	**0**	**46.7**	**0**	**1**	**0**	**1**	**-**	**0.64**	**0.35**	**0**

TABLE 7.4: Comparison of the error metrics in Scenario 6. FM1 is the FMeasure1. The < symbol means that the value is lower than 0.001.

CAMNEP methods detected tFP values until half of the scenario, and then they had some tTP. However, the tTPs were not enough to improve the FMeasure1 significantly.

7.5.4 Comparison of Results in Scenario 8

In this scenario, the botnet contacted a lot of different C&C hosts with Chinese-based IP addresses and received large amounts of encrypted data. It also scanned and cracked

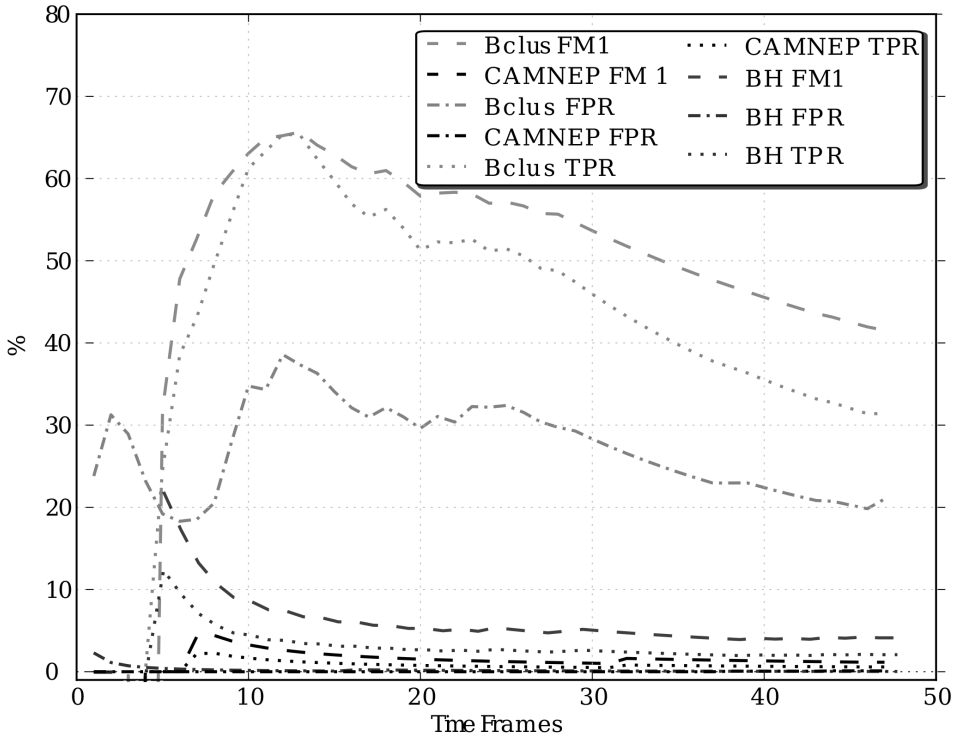

FIGURE 7.5: Comparison of the running error metrics for Scenario 2. FM1 is the FMeasure1.

the passwords of machines using the DCERPC protocol both on the Internet and on the local network for 19 hours.

The error metrics for this scenario can be seen in Table 7.5. The AllPositive algorithm had the best FMeasure1. Six algorithms were better than BClus, who had an FMeasure1 of 14% and a FPR of 30%. The CAMNEP algorithm had a FMeasure1 of 8% and a very low FPR. The BotHunter algorithm could not detect a single TP, so it was not possible to compute its FMeasure1. This Table shows that the CCDetector method achieved a TPR of 0 and an FPR of 0.4%. This scenario was difficult for all the methods, since none of them had an FMeasure1 of more than 14%. According to our analysis, this scenario only has one good and periodic C&C channel. This channel sent a large amount of flows. The CCDetector method was not able to detect this C&C channel because none of the trained models matched its behavior. However, there were some False Positives in this capture, which means that some normal models were misdetected. This is an example of a capture were the CCDetector was not trained with the correct models.

The simplified comparison for this scenario is shown in Figure 7.8. The BClus method had a low TPR of about 10%, however it was more than twice of CAMNEPs TPR value. The TNR value was near 60% for BClus and near 90% for CAMNEP. The FPR

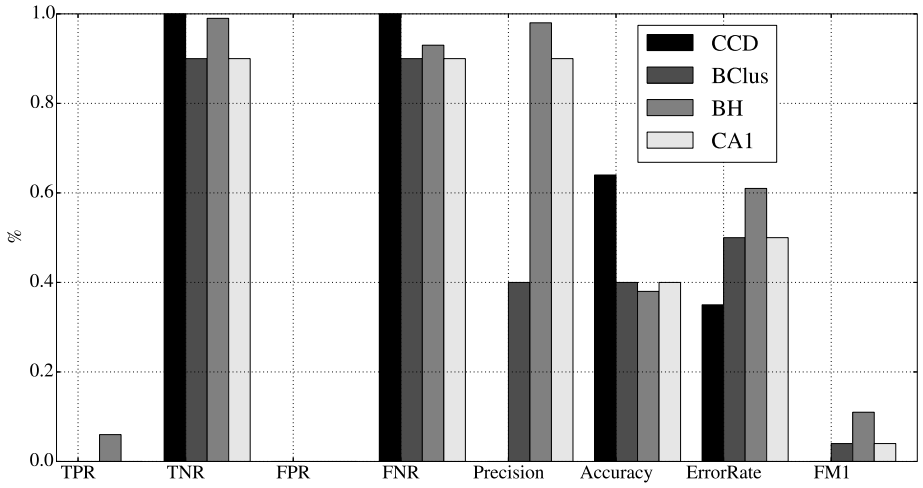

FIGURE 7.6: Simplified comparison of the error metrics for the BClus, CCDetector, CAMNEP and BotHunter algorithms on Scenario 6. FM1 is the FMeasure1.

of BClus was high, near 30%, however it had a FMeasure1 value that is twice the value for CAMNEP. Despite that the CCDetector method did not obtained a good FMeasure value, the Figure shows that it has the best Accuracy and the best ErroRate among all the methods.

The inner workings of the algorithms can be seen in the running metrics shown in Figure 7.9. The CCDetector method is not included in this plot because it was executed in a slightly different bidirectional dataset. This Figure shows that none of the error metrics exceeded 40%. It means that this scenario was very difficult for the methods. Until the fiftieth time frame both BClus and CAMNEP TPR values grew at almost the same rate. However, after that, the BClus method grew a little faster. The FPR of the BClus method was very high almost from the start of the scenario. The BotHunter algorithm had very low measurements during the whole scenario.

7.5.5 Comparison of Results in Scenario 9

In this scenario, ten host were infected using the same Neris botnet as in scenario 1 and 2. For five hours, more than 600 SPAM mails were successfully sent. The error metrics for this scenario can be seen in Table 7.6. The AllPositive algorithm had an FMeasure1 of 66%. The BClus algorithm had an FMeasure1 of 25%, a FPR of 20% and a TPR of 10%. The CAMNEP algorithm had a FMeasure1 of 17%, a very low FPR and a very low TPR. The BotHunter algorithm had a low FMeasure1 of 3% and high FNR of 98%. This Table also shows that the CCDetector method achieved the best FMeasure1 with a value of 54% and a FPR of 0.4%. This scenario is special because it includes the traffic

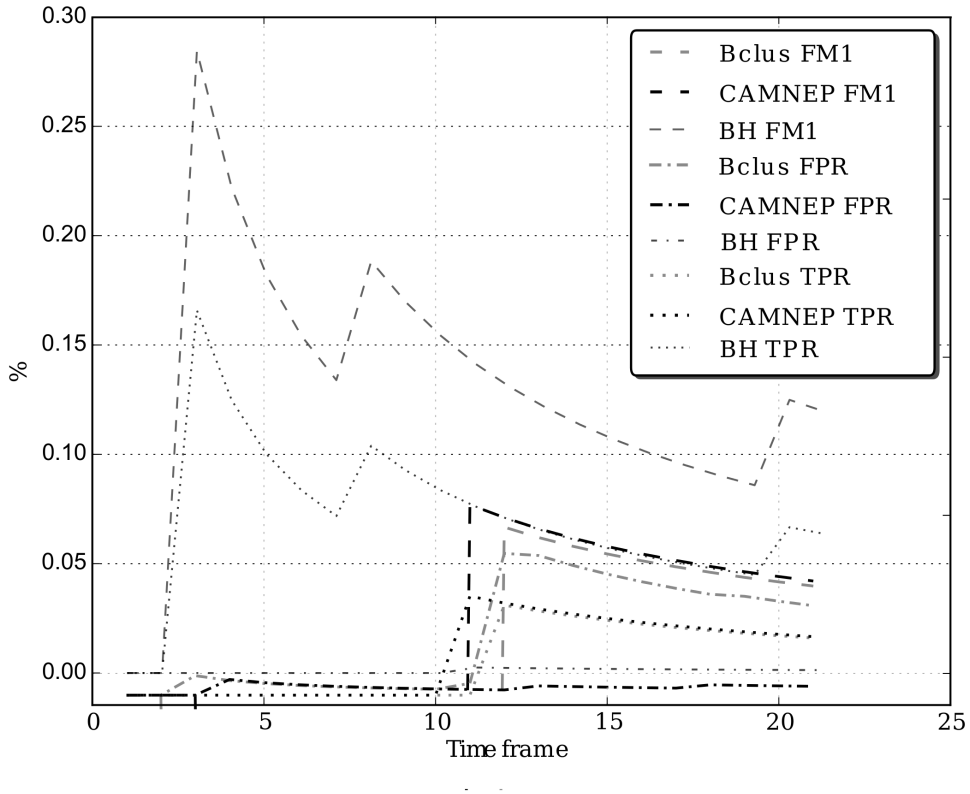

FIGURE 7.7: Comparison of the running error metrics for Scenario 6. FM1 is the FMeasure1.

of ten bots at the same time and therefore there is a large amount of botnet traffic to detect. This impacted all the methods, as it can be seen in the TPR of the CCDetector method, which had a value of 38%.

The simplified comparison for this scenario is shown in Figure 7.10. It can be seen that the TPR value for the BClus method was almost twice the value for CAMNEP. Also, the FPR value of BClus was 40 times larger than the CAMNEP value. However, the FMeasure1 value of CAMNEP was almost 70% the value of BClus.

The inner workings of the algorithms can be seen in the running metrics shown in Figure 7.11. The CCDetector method is not included in this plot because it was executed in a slightly different bidirectional dataset. Almost from the start of the scenario, the TPR and FMeasure1 of the BClus and CAMNEP methods grew fast. However, after the twentieth time frame, both FMeasure1 values started to decrease. The FPR value of BClus was relatively low compared to the previous scenarios. The BotHunter algorithm presented very low values during the whole scenario despite that there were ten bots being executed.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
AllPo	233.4	0	230	0	1	0	1	0	0.5	0.5	0.4	0.67
Fs1	74.4	220.7	9.2	159	0.3	0.9	<	0.6	0.8	0.6	0.3	0.46
Ko1	70.7	228.9	1.08	162.7	0.3	0.9	<	0.6	0.9	0.6	0.3	0.46
Ko1.5	68.4	229.3	0.6	164.9	0.2	0.9	<	0.7	0.9	0.6	0.3	0.45
Ko2	53.4	229.5	0.4	180	0.2	0.9	<	0.7	0.9	0.6	0.3	0.37
Fs1.5	54.6	222.7	7.2	178.8	0.2	0.9	<	0.7	0.8	0.5	0.4	0.36
Fs2	28	224.9	5	205.3	0.1	0.9	0.02	0.8	0.8	0.5	0.4	0.21
BClus	**23.5**	**152.8**	**77.1**	**209.9**	**0.1**	**0.6**	**0.3**	**0.8**	**0.2**	**0.3**	**0.6**	**0.14**
Fd1	15.6	196.1	33.8	217.7	<	0.8	0.14	0.9	0.3	0.4	0.5	0.1
Fd1.5	14.1	203.9	26	219.2	<	0.8	0.1	0.9	0.3	0.4	0.5	0.10
CA1	**10.8**	**229.8**	**0.1**	**222.6**	**<**	**0.9**	**<**	**0.9**	**0.9**	**0.5**	**0.4**	**0.08**
Fd2	11.6	209.8	20.1	221.8	<	0.9	<	0.9	0.3	0.4	0.5	0.08
X1	8.5	229.7	0.2	224.9	<	0.9	<	0.9	0.9	0.5	0.4	0.07
Mi1	4.6	213.1	16.8	228.7	<	0.9	<	0.9	0.2	0.4	0.5	0.03
Xd1.5	3.9	194.1	35.8	229.5	<	0.8	0.1	0.9	<	0.4	0.5	0.02
Xd2	3.9	194.4	35.5	229.5	<	0.8	0.1	0.9	<	0.4	0.5	0.02
Xd1	3.9	193.5	36.4	229.5	<	0.8	0.1	0.9	<	0.4	0.5	0.02
Mi1.5	1	219.2	10.7	232.4	<	0.9	<	0.9	<	0.4	0.5	0.008
X1.5	0.5	230	0	232.8	<	1	0	0.9	1	0.4	0.5	0.005
CCD	**0**	**674.8**	**30.5**	**306.2**	**0**	**0.95**	**0.04**	**1**	**0**	**0.66**	**0.33**	**-**
BH	**0**	**229**	**0.11**	**309**	**0**	**0.99**	**<**	**1**	**0**	**0.42**	**0.57**	**-**

TABLE 7.5: Comparison of the error metrics in Scenario 8. FM1 is the FMeasure1. The < symbol means that the value is lower than 0.001.

7.6 Conclusions

The CCDetector method showed a better performance in all the Scenarios for which it had the proper detection models. This means that the trained models could generalize well, since the training and testing scenarios do not share the same type of botnets. The contrast with the rest of the methods was very large, although more testing should be done to obtain a conclusion about the differences between the methods. It may be possible that with the correct detection models, the CCDetector method is able to achieve better results in all the experiments. The BClus method shows good results in all the experiments, but its FPR was still a little bit high for a real implementation. The CAMNEP method had a low FPR during most of the scenarios but at the expense of a very low TPR. The results of the BotHunter in the scenario 6 suggest that it may be still useful to have a statical method to blacklist known malicious IP addresses

The comparison of the BClus and CCDetector methods with the two third-party detection methods greatly helped to improve our research. The comparison showed how each method worked an a real dataset, highlighting the difficulties of dealing with different types of traffic. It was very helpful to analyze the conditions under which the methods were not optimal, and more importantly why the methods were having errors.

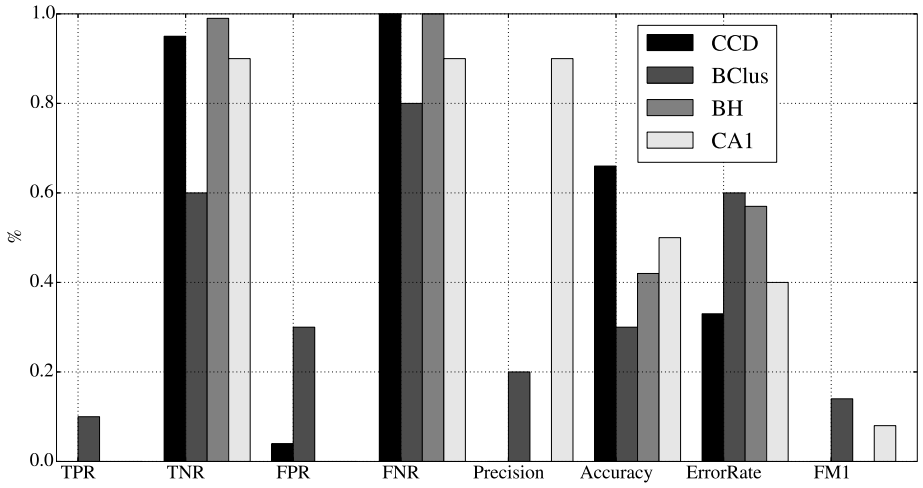

FIGURE 7.8: Simplified comparison of the error metrics for the BClus, CCDetector, CAMNEP and BotHunter algorithms on Scenario 8. FM1 is the FMeasure1.

It was very important for the comparison to use a large dataset with real labels. Despite not having a great amount of different botnets, it helped us noticing which phases of the botnet behavior were more easy to detect and how can the dataset be improved.

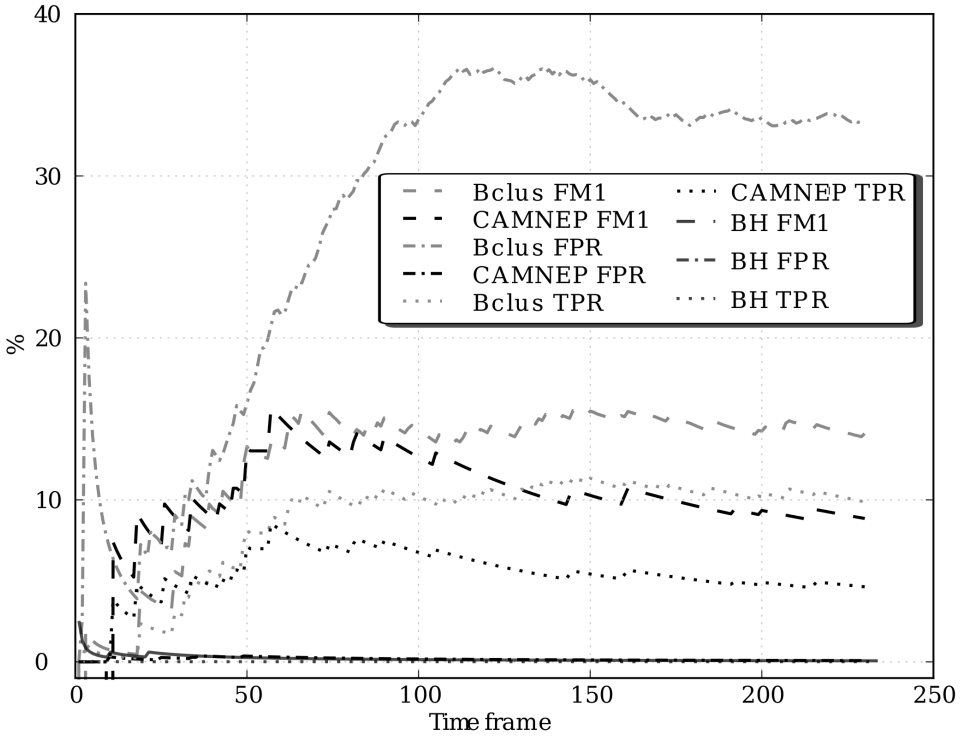

FIGURE 7.9: Comparison of the running error metrics for Scenario 8. FM1 is the FMeasure1.

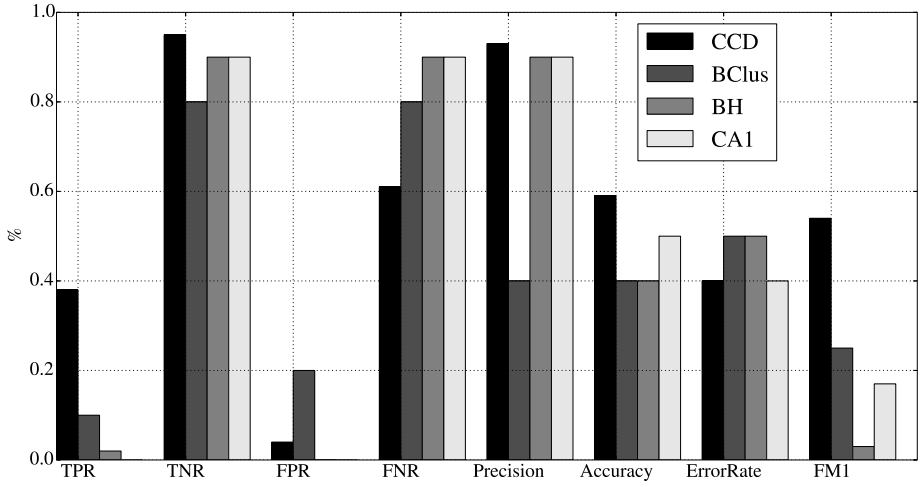

FIGURE 7.10: Simplified comparison of the error metrics for the BClus, CCDetector, CAMNEP and BotHunter algorithms on Scenario 9. FM1 is the FMeasure1.

Name	tTP	tTN	tFP	tFN	TPR	TNR	FPR	FNR	Prec	Acc	ErrR	FM1
CCD	**173.6**	**254**	**12**	**274.4**	**0.38**	**0.95**	**0.04**	**0.61**	**0.93**	**0.59**	**0.4**	**0.54**
AllPo	58.3	0	58	0	1	0	1	0	0.5	0.5	0.4	0.66
Fd1	21.7	47.1	10.8	36.6	0.3	0.8	0.1	0.6	0.6	0.5	0.4	0.47
Fd1.5	17.7	49.2	8.7	40.6	0.3	0.8	0.1	0.6	0.6	0.5	0.4	0.41
Fd2	13.9	50.9	7.01	44.4	0.2	0.8	0.1	0.7	0.6	0.5	0.4	0.35
Xd1	10.3	48.5	9.4	48.0	0.1	0.8	0.1	0.8	0.5	0.5	0.4	0.26
BClus	**10.1**	**46.4**	**11.5**	**48.2**	**0.1**	**0.8**	**0.2**	**0.8**	**0.4**	**0.4**	**0.5**	**0.25**
Fs1	8.3	55.2	2.7	50	0.1	0.95	<	0.8	0.7	0.5	0.4	0.23
Fs1.5	7.8	55.7	2.2	50.4	0.1	0.9	<	0.8	0.7	0.5	0.4	0.23
Xd1.5	7.04	53.3	4.6	51.3	0.1	0.9	<	0.8	0.6	0.5	0.4	0.2
CA1	**5.5**	**57.7**	**0.2**	**52.8**	**<**	**0.9**	**<**	**0.9**	**0.9**	**0.5**	**0.4**	**0.17**
Fs2	4.3	56.6	1.3	54	<	0.9	<	0.9	0.7	0.5	0.4	0.13
X1	2.8	56.9	1.09	55.5	<	0.9	<	0.9	0.7	0.5	0.4	0.09
Mi1	2.9	47.3	10.6	55.4	<	0.8	0.1	0.9	0.2	0.4	0.5	0.08
CA1.5	2.3	57.8	0.1	55.9	<	0.9	<	0.9	0.9	0.5	0.4	0.07
Mi1.5	2.2	51.3	6.6	56.1	<	0.8	0.1	0.9	0.2	0.4	0.5	0.06
BH	**1.76**	**57.9**	**0.06**	**86.9**	**0.02**	**0.9**	**<**	**0.9**	**0.9**	**0.4**	**0.5**	**0.03**
X1.5	0.9	57.3	0.6	57.4	<	0.9	<	0.9	0.6	0.5	0.4	0.03
Mi2	0.4	57.2	0.7	57.9	<	0.9	<	0.9	0.3	0.4	0.5	0.01
Ko1	0.2	57.8	0.1	58.1	<	0.9	<	0.9	0.6	0.4	0.5	0.008
Le1	0.1	57.2	0.7	58.1	<	0.9	<	0.9	0.1	0.4	0.5	0.006
X2	0.1	57.9	0.05	58.2	<	0.9	<	0.9	0.6	0.4	0.5	0.004
Ko1.5	0.06	57.9	0.03	58.3	<	0.9	<	0.9	0.6	0.4	0.5	0.002
Lv1	0.04	57.6	0.3	58.3	<	0.9	<	0.9	0.1	0.4	0.5	0.001
T1	0.04	58	0	58.3	<	1	0	0.9	1	0.4	0.5	<
CA2	0.04	58	0	58.3	<	1	0	0.9	1	0.4	0.5	<
Le1.5	0.03	57.7	0.2	58.3	<	0.9	<	0.9	0.1	0.4	0.5	<
T1.5	0.02	58	0	58.3	0	1	0	1	1	0.4	0.5	<
Ko2	0.02	57.9	0.01	58.3	0	1	0	1	0.5	0.4	0.5	<

TABLE 7.6: Comparison of the error metrics in Scenario 9. FM1 is the FMeasure1. The < symbol means that the value is lower than 0.001.

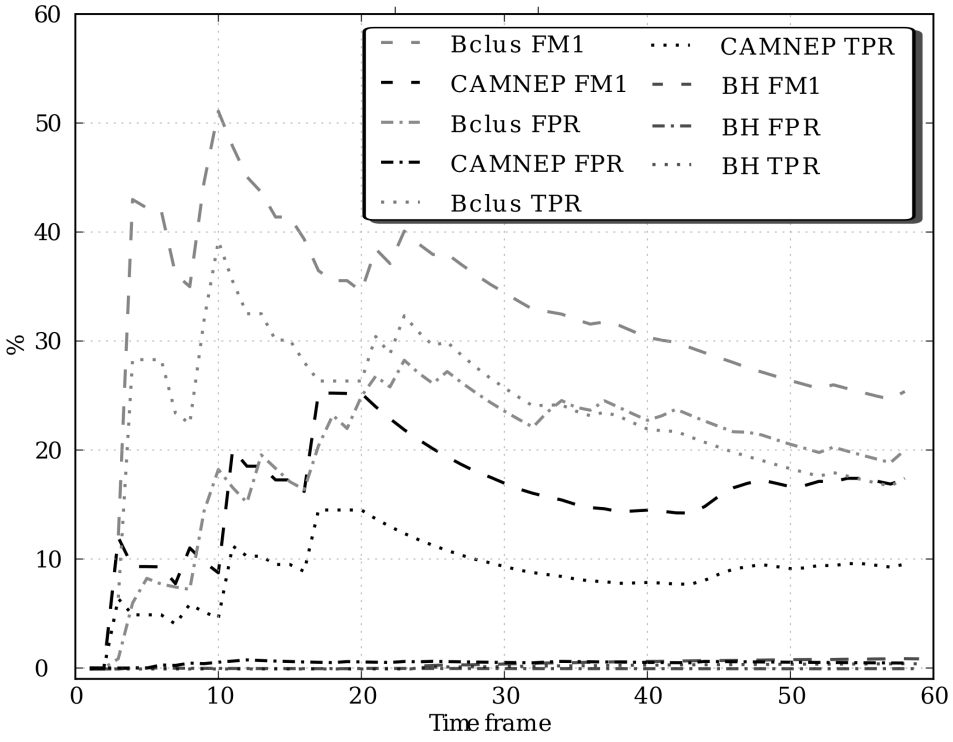

FIGURE 7.11: Comparison of the running error metrics for Scenario 9. FM1 is the FMeasure1.

Chapter 8

A Deep Description of our Ground-Truth Datasets

8.1 Introduction

The datasets used for this thesis were continually growing and improving. They started as some botnet captures in a home computer and ended up being a Malware Capture Facility Project. In this project we obtained a large amount of ground-truth captures that include dozens of botnets, normal captures, background data and attacks. All the captures were made using different topologies of computers and networks.

This Chapter describes each of the datasets used for the different detection methods presented in the thesis. We describe their designs and goals, the malware used, the behavior analyzed and their publication. The motivations to capture each dataset were different, so were their properties and the results obtained with them.

The more important lesson we learned while capturing our datasets is that without the proper Background traffic (i.e. unknown traffic) and Normal traffic, the utility of a Botnet dataset is seriously diminished, if it is not completely lost. When the research is done only with botnet data, the scope of what is *malicious* or normal is biased. We need to have Normal and Background data to put the botnet data into context and to have a better idea of how to avoid detection errors. It is true that sometimes we need to analyze only the botnet data to find out their best features, but at all times we should compare our findings with the three types of traffic to test our hypotheses. The next Sections describe the datasets used by each one of our detection methods.

Name	Duration	Unique Flows
Botnet1	11h:12m:29s	37,389
Botnet2	00h:20m:30s	21,124
Botnet3	10h:11m:33s	34,117
Non-botnet1	00h:10m:18s	45,148
Non-botnet2	03h:28m:14s	1,517
Non-botnet3	04h:19m:36s	976

TABLE 8.1: Labeled network data captures details

8.2 Dataset for the SimDetect Detection Method

The SimDetect detection method was the first method that we created and it is described in Chapter 3. In this method we used a dataset consisting of three botnet captures and three non-botnet captures. All the six captures were taken using three different testsbeds. The testbeds were used to acquire 92,630 unique botnet flows and more than 47,500 non-botnet flows. Table 8.1 shows each of the captures done in this dataset along with their duration and number of unique flows.

The first test bed was composed of a Linux computer virtualizing a Microsoft Windows XP SP3 operating system using VirtualBox. The VM (Virtual Machine) connected to the Internet through a DHCP (Dynamic Host Configuration Protocol) enabled home DSL router. The capture of packets was performed in the host Linux box with certain filters applied; assuring that only the Windows packets were captured. This testbed was created because some botnets usually target this type of home computers, looking for its high bandwidth and personal sensitive information [115].

The second testbed was composed of 18 Windows hosts in an internal sub-network of a University campus connected by a network hub. The network was monitored while five hosts were infected by hand. An extra computer using Linux was used for capturing the packets.

The third testbed was composed of one Linux computer connected to a University Campus WiFi Network. The WiFi network consisted in 40 normal computers. We captured the traffic of the whole WiFi network. We ensure that the traffic from this testbed was non-botnet by manually inspecting it and also scanning it with the Snort NIDS. The Non-botnet traffic includes SMTP mail, web sites with AJAX updates, edonkey like protocols, torrent protocols, operating system updates, and web traffic using several Google online tools.

With these testbeds we captured six different datasets. The first capture is called *Botnet1* and it was done by executing the *Virut* botnet in the first testbed. This botnet is an IRC-based variant capable of giving total control of the computer to the remote botmaster.

The second botnet capture is called *botnet2* and it was obtained by executing a botnet called *Agobot*. This botnet is web controlled and designed to send spam, self update and perform malicious actions.

The third botnet capture is called *botnet3* and it was obtained by executing a botnet called *Rbot*. This is a highly configurable IRC controlled bot used to gain access to infected computers in several different ways, including password cracking, SMB (Server Message Block) shares access and exploiting software vulnerabilities. This bot can be instructed to download and execute files, perform denial of service (DoS) attacks or log keystrokes among other attacks.

The three non botnet captures are called *non-botnet1*, *non-botnet2* and *non-botnet3*, and they were all taken using the third testbed. The capture *non-botnet1* consists of a port scanning activity using the nmap port scanning software. A port scanning was included because it represents one of the most common network attacks. Using nmap version 5.35DC18, we performed a TCP SYN scan of the 65,536 remote ports of one destination computer without changing the default timing behavior of nmap. The *non-botnet2* and *non-botnet3* captures are normal captures of the university network.

All the test beds in the SimDetect dataset were validated by ensuring a clean experimental setting. It means that all the Virtualized Windows were freshly installed, scanned with antivirus products and its traffic analyzed both with Snort NIDS (Network Intrusion Detection System) [116] and by hand. For the malware captures it means that malware binaries checked using VirusTotal [61] and the EUREKA! automated Malware Binary Analysis Service [62]. Also a manual packet analysis was done to verify the malware activity of *botnet1* and *botnet3* captures using the Wireshark [117] tool.

8.2.1 Publication of the SimDetect Dataset

To allow others reproduce the method, these datasets were published in the Malware Capture Facility Project on the following web pages:

- Botnet1: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-90/
- Botnet2: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-91/
- Botnet3: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-92/

8.3 Dataset for the BClus Detection Method

The dataset of the SimDetect method, described in the previous Section 8.2, was small and short. We quickly figured it out that we need a more robust dataset with more and better botnets. During the design of the BClus method we started a collaboration with the Czech Technical University to make a common dataset for the comparison of our algorithms.

In order to make a good comparison, a good dataset is needed. According to [10, 118], a good dataset should be representative of the network were the algorithms are going to be used. This means that it should have botnet, normal and background labeled data, that the balance of the dataset should be like in a real network (usually the percentage of botnet data tend to be small), and that it should be representative of the type of behaviors seen on the network. The difficulties to obtain such a dataset are discussed in Shiravi et al. [10] and the importance of these characteristics are discussed in Rossow et al. [9].

Due to the absence of a public botnet dataset with the characteristics needed, the collaboration with CTU ended with the creation of a new dataset that complies with the following design goals:

- Must have real botnets attacks and not simulations.
- Must have unknown traffic from a large network.
- Must have ground-truth labels for training and evaluating the methods.
- Must include different types of botnets.
- Must have several bots infected at the same time to capture synchronization patterns.
- Must have NetFlow files to protect the privacy of the users.

The topology used to create the dataset consisted in a set of virtualized computers running the Microsoft Windows XP SP2 operating system on top of a Linux Debian host. At the time of designing the topology, the Windows XP SP2 was the most used operating system by the malware. Each virtual machine was being bridged into the University network. Figure 8.1 shows a diagram of the testbed. The traffic was captured both on the Linux host and on one of the University routers. The traffic from the Linux host was exclusively composed of botnet traffic and was used for labeling purposes.

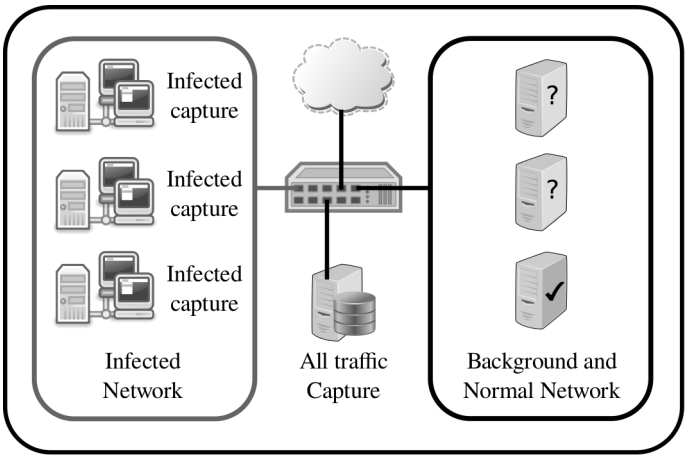

FIGURE 8.1: Testbed network topology.

The traffic from the University router was used to create the final dataset. The tool used to capture the traffic was tcpdump [75].

The next Subsections describe each of the captures, its design principles, the preprocessing of the dataset, the assignment of labels, the separation in training and testing and the publication of the dataset.

8.3.1 Design of the Botnet Scenarios

A botnet scenario is a particular infection of the virtual machines using a specific malware. Thirteen of these scenarios were created for this method, and each of them was designed to be representative of some malware behavior. The main characteristics of the scenarios and their behaviors are shown in Table 8.2. This table describes if they used IRC, P2P or HTTP protocols, if they sent SPAM, did Click-Fraud, port scanned, did DDoS attacks or if they were custom compiled. The features related with the network traffic of each scenario are shown in Table 8.3. It presents the size, duration, number of packets, number of flows, number of bots and bot family.

The network topology used to make the captures had a bandwidth control mechanism. However, the traffic going out to the Internet was not filtered. This decision may seem controversial [9], but it was taken with the explicit determination of capturing real attacks. We believe that the best way to study and model an attack is to capture real attacks. The next Subsection describes how these scenarios were preprocessed to obtain a more usable dataset.

Id	IRC	SPAM	CF	PS	DDoS	P2P	US	HTTP	Note
1	✓	✓	✓						
2	✓	✓	✓						
3	✓			✓			✓		
4	✓				✓		✓		UDP and ICMP DDoS.
5		✓		✓				✓	Scan web proxies.
6				✓					Proprietary C&C. RDP.
7								✓	Chinese hosts.
8				✓					Proprietary C&C. Net-BIOS, STUN.
9	✓	✓	✓	✓					
10	✓				✓		✓		UDP DDoS.
11	✓				✓		✓		ICMP DDoS.
12						✓			Synchronization.
13		✓		✓				✓	Captcha. Web mail.

Table 8.2: Characteristics of the botnet scenarios. (CF: ClickFraud, PS: Port Scan, US: Compiled and controlled by us.)

Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	11,231,035	52GB	Neris	1
2	4.21	71,851,300	7,037,972	60GB	Neris	1
3	66.85	167,730,395	15,202,061	121GB	Rbot	1
4	4.21	62,089,135	4,238,045	53GB	Rbot	1
5	11.63	4,481,167	7,710,910	37.6GB	Virut	1
6	2.18	38,764,357	2,579,105	30GB	Menti	1
7	0.38	7,467,139	454,175	5.8GB	Sogou	1
8	19.5	155,207,799	11,993,935	123GB	Murlo	1
9	5.18	115,415,321	8,087,513	94GB	Neris	10
10	4.75	90,389,782	5,180,852	73GB	Rbot	10
11	0.26	6,337,202	40,836	5.2GB	Rbot	3
12	1.21	13,212,268	1,262,790	8.3GB	NSIS.ay	3
13	16.36	50,888,256	6,425,345	34GB	Virut	1

Table 8.3: Amount of data on each botnet scenario.

8.3.2 Dataset Preprocessing

After capturing the packets using the pcap format, the capture files of the 13 scenarios in this dataset were preprocessed and converted to a common format for the detection methods. The format selected was the NetFlow file standard [94], which is considered the ad-hoc standard for network data representation. The conversion from pcap files to NetFlow files was done in two steps using the Argus software suite [90]. First, the *argus* tool was used to convert each pcap file into an bidirectional Argus binary storage file. The exact configuration of argus is published with each scenario in the web page of the Malware Capture Facility Project. Second, the *ra* Argus client tool was used to convert each Argus binary storage file into a NetFlow text file. This was done by specifying the desired output fields in the ra configuration. The ra configuration is also published with each scenario. These final NetFlow files were composed of the following fields: *Start Time*, *End Time*, *Duration*, *Source IP address*, *Source Port*, *Direction*, *Destination IP address*, *Destination Port*, *State*, *SToS*, *Total Packets* and *Total Bytes*.

8.3.2.1 Ground-truth Labels Assignment

The assignment of ground-truth labels is a very important part of the dataset creation process [119]. However, it can be complex and difficult to do [72]. A wrongly assigned label might produce unreliable results.

Our labeling strategy assigns three different labels in the capture files: background, botnet and normal. Background means unknown traffic, Normal are verified normal computers and botnet are the known infected machines. The assignment priority for the labels is the following:

1. Assign the Background label to the whole traffic.
2. Assign the Normal label to the traffic that matches certain filters.
3. Assign the Botnet label to all the traffic that comes from or to any of the known infected IP addresses.

The filters used to assign normal labels were created from the known and controlled computers in the network, such as routers, proxies, switches, our own computers in the laboratory, etc. The distribution of labels on each experiment is shown in Table 8.4. It can be seen that most of the traffic was labeled as Background. This majority class may add a natural bias to the dataset, however one of the ways to avoid this is to capture a large dataset, as it was stated in by Kotsiantis et al. [120].

Id	Background	Botnet	Normal
1	10,124,854 (95.40%)	94,972 (0.89%)	392,433 (3.69%)
2	6,071,419 (95.59%)	54,433 (0.85%)	225,336 (3.54%)
3	14,381,899 (94.60%)	75,891 (0.49%)	744,270 (4.89%)
4	3,895,469 (91.91%)	6,466 (0.15%)	336,103 (7.93%)
5	416,267 (91.37%)	2,129 (0.46%)	37,144 (8.15%)
6	2,031,967 (94.12%)	4,927 (0.22%)	121,854 (5.64%)
7	425,611 (93.71%)	293 (0.06%)	28,270 (6.22%)
8	11,451,205 (95.47%)	12,063 (0.10%)	530,666 (4.42%)
9	6,881,228 (90.22%)	383,215 (5.02%)	362,594 (4.75%)
10	4,535,493 (87.54%)	323,441 (6.24%)	321,917 (6.21%)
11	119,933 (29.33%)	277,892 (67.97%)	11,010 (2.69%)
12	119,933 (29.33%)	277,892 (67.97%)	11,010 (2.69%)
13	1,218,140 (93.76%)	21,760 (1.67%)	59,190 (4.55%)

TABLE 8.4: Distribution of labels for each scenario in the dataset of the BClus method.

8.3.2.2 Dataset Separation into Training, Testing and Cross-validation

To correctly create the classification models used in the BClus and CCDetector methods, we first needed to separate the dataset into training, cross-validation and testing. For the CAMNEP method the training was done using only the first 25 minutes of each scenario. The separation criteria of the scenarios was carefully evaluated, because the following constrains must be met:

- The training and cross-validation datasets should be approximately 80% of the dataset.
- The testing dataset should be approximately 20% of the dataset.
- None of the botnet families used in the training and cross-validation dataset should be used in the testing dataset. This ensures that the methods can generalize and detect new behaviors.

Since each scenario has multiple features (such as duration and size), it was not clear which of these features should be used for the 80%-20% separation criteria of the datasets. It is not the same to have an 80% of the amount of packets than of the amount of bytes. We made the separation by carefully selecting the scenarios so that the 80% of the following features are considered: the *Duration in minutes*, the *Number of clusters*, the *Number of NetFlows* and the *Number of aggregated NetFlows*. The final separation of the scenarios for the datasets is shown in Table 8.5. The problem of the imbalanced amount of labels on each dataset was reduced, as stated in Kotsiantis et al. [120], by carefully selecting the training and testing datasets. Also, as the majority label is *Background*, the bias toward the majority class reported in Li et al. [66] is avoided.

Scenario	Dataset
1,2,6,8,9	Testing
3,4,5,7,10,11,12,13	Training and cross-validation

TABLE 8.5: Dataset Separation into Training, Testing and Cross-validation for the BClus and CCDetector methods.

8.4 Publication of the BClus Dataset

These thirteen datasets were published and made available in our Malware Capture Facility Project. All the files can be downloaded, including the pcap file that has the botnet traffic. The only exceptions, for privacy issues, are the pcap files that contain the normal and background data. However, all the Argus and NetFlows files are published with the labels. The files can be found here:

- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-42/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-43/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-44/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-45/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-46/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-47/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-48/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-49/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-50/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-51/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-52/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-53/
- HTTPS://MCFP.FELK.CVUT.CZ/PUBLICDATASETS/CTU-MALWARE-CAPTURE-BOTNET-54/

8.5 Dataset for the CCDetector Detection Method

The dataset used for the creation of the CCDetector method and its comparison was the same that for the BClus method. We decided to use the same one because it is a large

and real dataset with background, normal and botnet labels. However, this dataset was re-analyzed because we acquired more experience in the detection of botnet behaviors. During the re-labeling of the dataset we assigned more than 1,471 new different labels. Each label identifies a specific botnet behavior. Only for the botnet traffic the dataset has 1,426 different labels. Our manual analysis of the dataset concluded that even the same C&C channel may show different behaviors, such as being active, idle or down. Despite that we label all the behaviors that we could recognize, we focused on the labeling of the C&C channels because they are a key component of the botnet infrastructure [121].

The original dataset consists in thirteen captures taken in a University Campus that include Botnet traffic, Normal traffic and Background traffic. The total size of the pcap files of this dataset is roughly 700GB. The goal of this section is to make a good description of the behaviors in the dataset in order to understand the results obtained and also to understand the type of traffic that the CCDetector method is able to detect.

The basic characteristics of this dataset was explained in the previous Section. Table 8.2 shows the basic details of each scenario: which application protocol was used, the type of attack done and the C&C protocol seen. The most common C&C was IRC, but there are also some P2P and HTTP channels. Sending SPAM seems to be the most common attack, followed by click-fraud. No scenario used fast-flux techniques. The type of traffic seen on each scenario vary greatly, not only because of the different botnets but also because of the different background traffic that was being captured at the same time. Table 8.3 shows the duration of each scenario, the number of packets, the number of flows, the size, the botnet used and the number of simultaneous infected bots. In some scenarios we used more than one bot to analyze their coordinated actions.

It should be noted that the biggest difference in the dataset with respect to the BClus comparison of Section 8.3 is that for the CCDetector method we used bidirectional flows instead of uni-directional flows. This means that our representation of the network traffic in this dataset is different to the one used for the BClus method, but the network traffic itself is the same. This change is reflected in the different number of flows in Table 8.6. As part of the description of the dataset the next Subsection explains the methodology for assigning the labels.

8.5.1 Assignment of ground-truth labels

A good and well designed dataset is important for training and testing, but more important are the labels assigned to the dataset and how the assignment was done. However, most botnet detection methods do not describe the label assignment process and seem to underrate its importance [8]. It is common to find detection methods that

Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	2,824,637	52GB	Neris	1
2	4.21	71,851,300	1,808,123	60GB	Neris	1
3	66.85	167,730,395	4,710,639	121GB	Rbot	1
4	4.21	62,089,135	1,121,077	53GB	Rbot	1
5	11.63	4,481,167	129,833	37.6GB	Virut	1
6	2.18	38,764,357	558,920	30GB	Menti	1
7	0.38	7,467,139	114,078	5.8GB	Sogou	1
8	19.5	155,207,799	2,954,231	123GB	Murlo	1
9	5.18	115,415,321	2,753,885	94GB	Neris	10
10	4.75	90,389,782	1,309,792	73GB	Rbot	10
11	0.26	6,337,202	107,252	5.2GB	Rbot	3
12	1.21	13,212,268	325,472	8.3GB	NSIS.ay	3
13	16.36	50,888,256	1,925,150	34GB	Virut	1

TABLE 8.6: Amount of data on each scenario of the CCDetector dataset. The amount of NetFlow is different from the BClus dataset.

consider all the background traffic as Normal and all the traffic coming from an infected machine as Botnet. This type of assignment can lead to labeling problems, for example labeling as normal the traffic that is coming out of an infected computer but that was generated by a not-infected process. The importance of labels and detection methods has been proven in [98], saying that if the training data is mis-labeled, some boosting algorithms fail to produce a model with an accuracy better than 50%.

For the creating and comparison of the CCDetector method we manually analyzed the traffic to better understand the behaviors and to label them accordingly. Instead of labeling each IP address as Botnet or Normal (as it was done in the BClus method), we labeled individual behaviors. Each behavior is related to one 4-tuple and each 4-tuple is related to one source IP address. Subsection 6.2.2 describes the concept of 4-tuple.

To find out the ground-truth C&C labels that are needed for the CCDetector method we manually analyzed the network traffic of each botnet with a set of tools that allow us to visualize the traffic: tcpdump [75], wireshark [117], tcpflow [122], oip [123] and malcom [124]. These tools helped us identify which flows were related to C&C activity based on its periodic behavior, based in the persistence of its connections, based on its encryption (or plain text if it was available), based on the information sent and on the relationship between the C&C traffic and the actions done by the botnet. With this information it was possible to assign very detailed labels to the traffic.

Once the behaviors were identified we used the *ralabel* tool of the Argus suite to add the labels into the Argus file. This allow us to permanently store the labels within the NetFlows and work with them as any other NetFlow field. The *ralabel* tool reads a rule-based configuration file with the logic of how to assign the labels to the binary Argus

file. The *ralabel* rules allow us to assign labels based on all the Argus fields, including the state of the TCP and UDP flows. The rules are applied from top to bottom, and ralabel stops searching once one rule matches.

Once a C&C channel is identified in the traffic, a label can be assigned. This label should be as detailed as possible to better separate different behaviors. The labels are assigned based on the following criteria:

- Identify the source IP address of the known bot or normal computer.
- Identify the destination IP address and destination port of the identified service.
- Differentiate if the flow is *from* the host or *to* the host.
  This is useful to distinguish real connections generated by the host from unknown connections from the Internet.
- Identify the network-layer protocol: UDP, TCP, ICMP or ARP.
- Identify if the connection is Established or an Attempt.
  Established connections and Attempts are tracked by Argus.

The text of the labels assigned include the following information, if possible:

- Identify if it is a C&C flow, Botnet flow, Normal flow or Background flow.
- Use an incremental numeric identifier for the type of C&C.
  Useful to differentiate different flows of the same C&C.
- Identify the 3rd layer protocol: TCP, UDP, ARP or ICMP.
- If possible identify the application-layer protocol: HTTP, HTTPS, etc.
- Have an identifier for the scenario number, e.g V1, V2.
  Useful to distinguish the flows of each capture when we put them together.
- Identify the general type of connection, such as encrypted C&C, custom encryption, Advertising, SPAM, etc.

An example rule for assigning a label to a C&C established connection coming from the IP 147.32.84.165 to IP address 97.74.144.110 and port 80 using the TCP protocol is:

```
filter="src host 147.32.84.165 and tcp and synack and dst host
 97.74.144.110 and dst port 80"
 label="From-Botnet-V44-TCP-CC100-HTTP-Custom-Encryption"
```

In this example, the *V44* text identifies the 3rd scenario and the text *CC100* means the 100th C&C channel identified with an HTTP protocol and Custom Encryption. The method used for determining if the traffic was encrypted is the same as in a previous paper [92]. Argus has several ways of identifying an established connection for TCP and UDP connections, in this example we use the keywords *synack* or *con*.

Another example of a rule used to identify a binary download using an HTTP established connection coming *from* a botnet but that is not related to any C&C is the following:

```
filter="src host 147.32.84.165 and tcp and synack and dst host 94.63.149.152
and dst port 80"
label="From-Botnet-V42-TCP-Established-HTTP-Binary-Download-1"
```

Another example of a rule for assigning a label to the Normal flows coming *from* our web server is the following:

```
filter="src host 147.32.87.36" label="From-Normal-Department-WebServer"
```

An example of a connection with flows coming *to* the web server and that is labeled as Background because we are not sure of their nature is the following:

```
filter="dst host 147.32.87.36" label="To-Background-Department-WebServer"
```

The following is an example of a Background rule assigned to one particular host that we identified but do not trust:

```
filter="host 147.32.80.102" label="Background-Jones-Host-Department"
```

The following rule is an example of a generic Background rule for established connections using the NTP protocol:

```
filter="udp and con and dst port 123" label="Background-UDP-NTP-Established-1"
```

The following is an sample rule for all the established TCP background flows:

```
filter="tcp and synack" label="Background-TCP-Established"
```

Finally, if no rule matches, the default is to assign a generic Background label:

Scenario	Botnet Labels	Normal Labels	C&C Labels	Background Labels
1	43	8	28	37
2	61	7	36	36
3	6	8	1	36
4	10	8	2	36
5	27	8	8	36
6	6	8	1	36
7	8	6	2	36
8	11	8	3	36
9	727	8	278	36
10	52	8	16	36
11	11	8	3	36
12	29	7	3	33
13	50	8	30	36

TABLE 8.7: Different unique ground-truth labels assigned on each scenario.

```
filter="" label="Background"
```

In the dataset we maintained a high confidence about the origin of the data. The source IP addresses for the Botnet rules (the IP addresses of the bots), were known because we infected them manually. The IP addresses of the Normal computers were manually selected from a small and verified group of known computers in the University. The rest of the computers in the laboratory, even our own servers and printers, were not considered normal because we could not guarantee that they were not attacked during the captures.

Using our large and verified group of rules we assigned the labels to each scenario in the dataset. The labels in the training captures were used to train the model and the labels in the testing captures were used for verification. Table 8.7 shows the amount of different unique labels assigned to each scenario. In this table it can be seen that there are around 36 different and unique labels for the Background traffic, that usually there are less C&C labels than Botnet labels, which is coherent, and that the amount of different normal labels is around 8. The Table also shows that the scenario 9 has a huge amount of botnet and C&C labels. This is because we infected 10 bots in that scenario.

After the assignment, each scenario ended with a different amount of labeled flows. Table 8.8 shows the amount of labeled flows on each scenario differentiated by the type of label. It can be seen that the amount of Normal flows fluctuates between 0.5% and 3.6% for all the scenarios and the amount of C&C flows fluctuates between 0.002% and 2.4%. This Table shows that capture 8 and 5 are the ones with more percentage of C&C flows and that capture 9 is the one with more absolute number of C&C flows.

Scen.	Total Flows	Botnet Flows	Normal Flows	C&C Flows	Background Flows
1	2,824,636	39,933(1.41%)	30,387(1.07%)	1,026(0.03%)	2,753,290(97.47%)
2	1,808,122	18,839(1.04%)	9,120(0.5%)	2,102(0.11%)	1,778,061(98.33%)
3	4,710,638	26,759(0.56%)	116,887(2.48%)	63(0.001%)	4,566,929(96.94%)
4	1,121,076	1,719(0.15%)	25,268(2.25%)	49(0.004%)	1,094,040(97.58%)
5	129,832	695(0.53%)	4,679(3.6%)	206(1.15%)	124,252(95.7%)
6	558,919	4,431(0.79%)	7,494(1.34%)	199(0.03%)	546,795(97.83%)
7	114,077	37(0.03%)	1,677(1.47%)	26(0.02%)	112,337(98.47%)
8	2,954,230	5,052(0.17%)	72,822(2.46%)	1,074(2.4%)	2,875,282(97.32%)
9	2,753,884	179,880(6.5%)	43,340(1.57%)	5,099(0.18%)	2,525,565(91.7%)
10	1,309,791	106,315(8.11%)	15,847(1.2%)	37(0.002%)	1,187,592(90.67%)
11	107,251	8,161(7.6%)	2,718(2.53%)	3(0.002%)	96,369(89.85%)
12	325,471	2,143(0.65%)	7,628(2.34%)	25(0.007%)	315,675(96.99%)
13	1,925,149	38,791(2.01%)	31,939(1.65%)	1,202(0.06%)	1,853,217(96.26%)

TABLE 8.8: Amount of labeled flows on each capture.

The final separation of the scenarios into training, validation and testing datasets was the same that the separation for the BClus method 8.5. This decision allow us to better compare our method with those previous results, since we test on the same scenarios. Table 8.5 shows which scenario is part of which type of dataset.

There is one more explanation that should be done about the label assignment process. It is related to how the same unique label can be used for several different rules when they are related to the same C&C channel behavior. This procedure effectively *join* two network behaviors as one. This is only done if the behaviors of the different 4-tuples are homogeneous, meaning that they are really similar for the experts. This type of rule aggregation is good because only one label is used for several rules that identifies the same behavior in the network. For example, the following rules from the same scenario were identified as part of the same C&C channel even if they were using a different destination IP address and port. They were assigned the unique label *From-Botnet-V1-TCP-CC108-Plain-HTTP*:

- filter="src host 147.32.84.165 and tcp and synack and dst host 123.126.51.33 and dst port 80"

  `State:33ccttttttCttttttttttttttttttCcttatccaaaaccbccccccccccccccccccccaaaaccccccccccacccccccccccccccccaaaccccccccccccccccCttacccaaccccccc(...)`

- filter="src host 147.32.84.165 and tcp and synack and dst host 72.20.15.61 and dst port 80"

  `State:33ctttctttttttstcccccaccccccccccccccccccccaccccccccccccccccccc`

```
ccccccccccccccccccccccccccccccccccccccccccccttcccccccccccccccccc(...)
```

- filter="src host 147.32.84.165 and tcp and synack and dst host 174.37.196.55 and dst port 80"

```
State:33ccccttttttttttttttttttttttttttttcttcccccccccccccccccccccccc
ccccccccacccccccccccccccccccccccccaaaaccccccaaccccccccccccaa(...)
```

8.5.2 Brief Analysis of the Fitness of the Dataset

The error metrics generated by a detection method depend not only on the method itself, but most importantly on the quantity and quality of labels assigned and on the content of the dataset. After assigning the labels in the previous Subsection, we briefly discuss here which is the content of our dataset and if our method should detect those botnets actions or not. The idea is that not every packet being sent by an infected machine should be detected as botnet or malware, but only those packets and flows that are representative of a botnet behavior.

Each scenario in the dataset is composed of several actions done by the botnet and several C&C behaviors. In the Appendix A.1 we make a detailed analysis and description of each C&C channel on each scenario, including an example of the packets and strings sent. We used that analysis to determine in this Subsection if each scenario is a good representative of a botnet behavior and if we expect our method to detected it or no. The summary of our analysis can be seen in Table 8.9. It shows that botnets usually had between 1 and 10 C&C channels and that some of these C&C channels are not a good representative of their behavior. This usually means that the data captured is not enough to show the pattern or that the C&C channel does not show a differentiate pattern in the network.

The scenario 1 has 4 verified C&C channels with a strong periodicity that are good representatives of the botnet behaviors and they should be detected by our method. The scenario 2 has 6 verified C&C channels with a strong periodicity (and some with a weak periodicity) that are good representatives of the botnet behaviors and should be detected. The scenario 3 has 1 verified C&C channel with a weak periodicity and several timeouts that is a not bad representative of an IRC behavior and should not be detected by our method. The scenario 4 does not have any C&C behavior that is a good representative of the botnet. The scenario 5 has 2 verified C&C channels, but they are a poor representation of the botnet behaviors, however, they are still C&C and should be detected. The scenario 6 has 1 verified C&C channel with a strong periodicity that is a good representative of the botnet behaviors and should be detected. The scenario

Scenario	# CC Channels	Represents botnet behavior?
1	4	Good
2	6	Some good, some fair
3	1	Fair
4	0	-
5	2	Poor
6	1	Good
7	1	Good
8	1	Good
9	50	Most good, some fair.
10	1	Poor
11	0	-
12	4	Poor
13	4	Good

TABLE 8.9: Summary of the CC channels and type of behaviors on each scenarios. The last column indicates if the behavior should be detected or not based by the CCDetector method.

7 has 1 verified C&C channel with a strong periodicity that is a good representative of the botnet behaviors and should be detected. The scenario 8 has 1 verified C&C channel with a strong periodicity that is a good representative of the botnet behaviors and should be detected. The scenario 9 has 50 verified C&C channels with a good periodicity. Most of them are good representatives of the botnet behaviors and some are a fair representation, but all of them should be detected. The scenario 10 has 1 verified C&C channel with almost no periodicity that is a poor representation of the botnet behaviors. However it should be detected because it is a IRC C&C channel. The scenario 11 does not have any good representative of a C&C channel because the only IRC-based C&C connection is so short that it only generated one flow. The scenario 12 has 4 verified C&C P2P connections, but they are not good representatives of the botnet behaviors because they are short. The scenario 13 has 4 verified C&C channels that are good representatives of the botnet behaviors.

8.6 Publication of the CCDetector Dataset

The links to download these datasets are the same as in Subsection 8.4. However the new labels in the bidirectional NetFlows can be downloaded from these subfolders:

- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-42/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-43/detailed-bidirectional-flow-labels/

- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-44/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-45/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-46/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-47/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-48/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-49/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-50/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-51/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-52/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-53/detailed-bidirectional-flow-labels/
- https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-54/detailed-bidirectional-flow-labels/

Moreover, a unique big file with all the thirteen datasets together was published and described here: http://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html

8.7 Analysis of Third-party Datasets that were Already Published

After describing our own datasets, this section offers an overview of the already published malware datasets in the literature and why there were not fitted to our purposes in this thesis.

To be useful in the creation of a detection method or in a comparision effort, a botnet datasets should have some specific characteristics. A good analysis of these characteristics has been done in Shiravi et al. [10] about the generation of datasets. It describes the properties that a dataset should have in order to be used for comparison purposes. The dataset used Shiravi et al. [10] includes an IRC-based Botnet attack[1], but the bot used for the attack was developed by the authors and therefore it may not represent a real botnet behavior. This dataset is currently unavailable, however it is advertised as ready for download.

The Protected Repository for the Defense of Infrastructure Against Cyber Threats (PREDICT) indexed three Botnet datasets[2] until May 16th, 2013. The first one is the *Kraken Botnet Sinkhole Connection Data* dataset, the second one is the *Flashback Botnet Sinkhole Connection Data* dataset and the third one is the *Conficker Botnet Sinkhole Connection Data* dataset. They were published as CSV text files, where each line is a one minute aggregation of the number of attempted connections of one IP address. Unfortunately, the aggregation method and the CVS format may not be suitable for comparisons with other proposals.

The CAIDA organization published a paper about the Sality botnet in Dainotti et al. [125] along with its corresponding dataset[3]. Unfortunately, the CSV text format of the dataset may not be suitable for every detection algorithm because the content of the dataset only includes the information required to reproduce the techniques described in the paper. CAIDA also published a dataset about the Witty Botnet in pcap format[4] and several datasets with responses to spoofed DoS traffic[5] and anomalous packets[6]. However, none of them are labeled.

A custom botnet dataset was created to verify five P2P botnet detection algorithms in Saad et al. [7]. Fortunately, this dataset was made public and can be downloaded[7]. The dataset is a mixture of two existing and publicly available malicious datasets and one non-malicious pcap dataset. They were merged to generate a new file. This was, at that time, the best dataset that can be downloaded for comparison purposes. Unfortunately, there is only one infected machine for each type of botnet, therefore no synchronization analysis can be done.

[1] http://www.iscx.ca/datasets
[2] https://www.predict.org/Default.aspx?tabid=104
[3] http://imdc.datcat.org/collection/1-06Y5-B=UCSD+Network+Telescope+Dataset+on+the+Sipscan
[4] http://www.caida.org/data/passive/witty_worm_dataset.xml
[5] http://www.caida.org/data/passive/backscatter_2008_dataset.xml
[6] http://www.caida.org/data/passive/telescope-2days-2008_dataset.xml
[7] http://www.isot.ece.uvic.ca/dataset/ISOT_Botnet_DataSet_2010.tar.gz

The Traffic Laboratory at Ericsson Research created a normal dataset that was used in Saad et al. [7] and described in Szabo et al. [126]. This normal dataset is publicly available[8]. It is composed of pcap traffic files that were labeled by means of one IP header option field. This is the only normal dataset that is labeled inside the pcap file.

A considerable amount of malware traffic in pcap format was published in the Contagio blog[9]. It contains, until August 2014, thirty three ATP pcap captures and sixty one crimenware pcaps. Each file contains the traffic of one malware without background traffic. Unfortunately, the captures are really short (mostly between 1 minute and 1 hour) and the traffic is not labeled. But since each scenario includes only one infected computer, it should be possible to label them.

Another dataset with malware logs and benign logs was collected in NexGinRC [127]. The malware logs are both real and simulated. The benign logs consist of 12 months of traffic. Unfortunately, the dataset is in CSV format, which may not be suitable for some detection algorithms because it does not have the same information as a NetFlow file or pcap file. Access to this dataset may be granted upon request[10].

The last dataset analyzed is currently created by the MAWI project described in Cho et al. [128]. It includes an ongoing effort to publish one of the most recent and updated background datasets to date. Its goal is to promote the research on traffic analysis and the creation of free analysis tools. However, the pcap files are not labeled, and therefore it is more difficult to use them for training or verification. There was an effort to label this dataset using anomaly detectors in Fontugne et al. [119]. The labels are not ground-truth, but may be useful to compare other methods.

A final summary of the previously described datasets is presented in Table 8.10. This table shows that, so far, no dataset includes Background, Botnet and Normal labeled data.

8.8 Malware Capture Facility Project

The vast majority of the botnet captures used in our thesis were done in our Malware Capture Facility Project (MCFP). This project has the goal of capturing and publishing long-term malware traffic. The captures usually last from one week to several months. We believe that only capturing long-term botnet traffic it is possible to understand and find the most important behaviors of the botnets.

[8] http://www.crysys.hu/~szabog/measurement.tar

[9] http://contagiodump.blogspot.co.uk/2013/04/collection-of-pcap-files-from-malware.html

[10] http://nexginrc.org/Datasets/DatasetDetail.aspx?pageID=24

Name	Available	Format	Background	Botnet	Normal	Labels
Shiravi et al. [10]	?	?	✓	✓	-	?
PREDICT	✓	CSV	-	✓	-	-
CAIDA	✓	CSV, pcap	-	✓	-	-
Saad et al. [7]	✓	pcap	✓	✓	-	✓
Szabo et al. [126]	✓	pcap	-	-	✓	✓
Contagio	✓	pcap	-	✓	-	✓
NexGinRC [127]	?	CSV	-	✓	✓	?
Cho et al. [128]	✓	pcap	✓	-	-	-

TABLE 8.10: Summary of available datasets

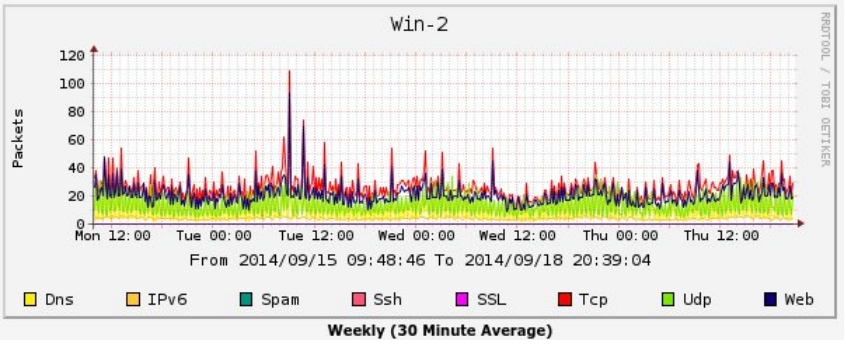

FIGURE 8.2: Example of the Cacti monitoring software used in the MCFP for controlling the malware traffic in real time.

The MCFP is running 20 virtual machines on a University and storing the traffic in pcap files. The malware binaries are selected from public and private sources and monitored using a Cacti [11] web server. Figure 8.2 shows an example of the Cacti monitoring software being used to control the traffic of botnets in real time. The Cacti software as well as the RRD files for each capture were customized to include the type of meta data about the traffic that we need. For example, we plot the SPAM traffic by searching for the established connections to the TCP ports 25 or 587. The MCFP help us obtain the necessary understanding of the botnet behaviors to create our detection methods.

So far the MCFP has obtained 92 botnet captures and more than 670GB of pcap data. These pcap files are processed to assign labels, obtain an Argus binary file, a NetFlow text file with labels and several complement files such as web logs, DNS data, etc. All this information is being published in its web site `http://mcfp.felk.cvut.cz`

[11] `http://www.cacti.net/`

Chapter 9

Conclusions

Botnets continue to be one of the most important threats to the Internet community. They are the main technology that supports the attacks of sponsored malware [129], the spying of organizations [130], the abuse of advertisement programs, the steal of private information, the steal of money and the persecution of political dissidents [131] among others. Moreover, these actions are so complex and diverse that it is known to be hard to detect them [52] [132]. However, despite their complexity, this thesis supports the idea that it may be possible to detect the behavioral patterns of botnets in the network.

In spite of all the efforts in the detection area, there is no unique solution that can protect from all the botnets attacks. At most, it is possible to detect some malware with antivirus software and to detect certain domains, URLs and IP using Threat Intelligence sources. But these protections defend mostly against a specific threat and can not keep up with the rapid development of new malware. Fortunately, the application of Machine Learning techniques to this problem has started to produce promising results and long-term solutions.

The goal of our thesis was to create a behavioral-based and network-based botnet detection method. We chose to work with network traffic because it is the best way to protect thousands of computers at the same time, and we chose to work with behavior because it seemed to be the best way to represent the complex actions of botnets. To achieve our final goal we planned and completed other sub-goals, such as to understand the botnet behaviors, to compare our methods with other techniques, to create a large botnet dataset and to publish our tools and algorithms as free software. We selected the free software license because too many detection methods were converted to private products and now they are unavailable to the vast majority of Internet users.

The plan to complete our goal had several steps. We created a malware capture facility to get good, labeled and real ground-truth malware traffic. We developed the SimDetect detection method to detect similarities in botnet traffic. We then learnt from our errors and developed the BClus method to automatically cluster and detect groups of botnet behaviors. And finally, with a better understanding of the botnet behaviors, we developed the CCDetector method that detects the behavioral patterns of the botnets C&C channels in the network by using a novel state-based model of the network traffic. The CCDetector method summarizes all what we have learnt about the detection of botnets during this thesis. Furthermore, the BClus and CCDetector methods were compared with other two third-party methods, verifying our results with state-of-the-art techniques.

The SimDetect method was a first attempt at analyzing the behavior of botnets by detecting similarities in the traffic. It separates the traffic in time windows, aggregates some features and clusters the aggregated features. The dataset used is composed of normal traffic, botnet traffic and manual attacks. The results were promising, since it was possible to cluster the botnet behavior quite accurately. However, this method had strong limitations. First it lacks an automatic way of detecting the botnet clusters. Second, the aggregation of features was too aggressive and, therefore, too much information was lost. Third, the dataset was too simple and lacked good background traffic. Our conclusion for this method is that it was good for learning and testing ideas but it lack some important features of a detection algorithm.

To solve the above mentioned issues and to improve the results we developed our second detection method, called BClus. We also created a much better, large and labeled botnet dataset. This method also separates the traffic in time windows, aggregates a better selected group of features and clusters the aggregated features. Since the dataset was labeled, the method was trained to learn how the botnet clusters looked like using a classification algorithm. The rules from the classification algorithm were used to detect similar clusters in unknown traffic. The BClus method generated good results that were compared with other two third-party detection methods using the same dataset. The BClus method achieved the best results in this comparison. However, it also had some limitations: it was difficult to generalize, the rules were difficult to merge with each other and most importantly, it was unclear which was the meaning of a botnet cluster. Our conclusion for this method is that it was a mayor step in the detection capabilities, producing better results. However, the results were still not good enough to be considered in a real environment.

After analyzing hundreds of botnet captures and identifying hundreds of botnet actions, we designed and implemented our third detection method, called CCDetector.

This method solves the issues of the previous methods by using a completely novel representation of the botnet actions. The CCDetector method can represent the behavioral pattern of any network connection by storing the transitions between its *states*. Each state is defined by some features of each network flow. Using the transitions between states the method can generate a chain of states that represents its behavior over time. This chain of states can be stored, shared and used for multiple tasks. The CCDetection method uses these chains of states to create a Markov Model of the behavior of the connection. These network behaviors may represent botnet traffic or also normal traffic. The CCDetector method uses these trained botnets traffic patterns to detect similar behaviors in unknown networks. To evaluate the method, the results were compared with other three detection methods, including our previous BClus method and other two third-party methods. The CCDetector method achieved very good results and proved suitable to detect new botnets in the network. Our conclusion for this method is that the behavioral-based state model accurately represents the botnet behaviors in the network, and that the detection model showed a very promising set of results. Both models may be a good advance in the detection of botnet traffic in the area.

An important part of our thesis has been the huge botnet dataset captured. With more than 800GB of botnet traffic, this labeled dataset was the basis to understand the botnets actions. Our deep study of the different botnets behaviors lead to a very detailed manual labeling process, that was paramount to the creation of our detection methods. This dataset is a huge improvement in the botnet detection area and therefore is public available to the community along with the analysis of each capture.

The most significance part of our work is the new state-based behavioral model of network traffic and the CCDetector detection method. The state-based model successfully embodies the network behaviors and opens new opportunities for representing and working with network actions. In the case of our CCDetector method it means that the botnet actions, and in particular botnets C&C channels, can be recognized with very error metrics. We hope that the free publication of our method and dataset would help the security community to create better detection methods.

The original contributions of our thesis are:

- A novel state-based model representation of the network behaviors that can be used on any type of traffic.
- A novel and free behavioral-based detection method called CCDetector that can accurately detect botnet C&C channels.
- A novel, large, real and public botnet dataset with background, normal and botnet labels.

The final conclusion of our thesis is that it is possible to model the behavioral patterns of botnets in the network and to use these models to detect unknown botnets.

9.1 Open Problems and Future Research Directions

There could be several implications of using our novel state-based model of network behaviors. The first implication is related to the detection of botnets, as shown by the CCDetector method. This method may allow any network administrator or normal Internet user to detect botnet behaviors in their network using a robust and free detection tool. A second implication may be the use of these models to work with normal traffic. It may be possible to precisely identify normal traffic as it was done with botnet traffic. The usage of the state-based models to identify normal traffic may have also an impact on the enforcement of network policies in organizations. This future research may allow to detect and stop other types of traffic, such as audio streaming, password bruteforcing, or P2P traffic. This may give the opportunity to, for example, separate normal P2P traffic from botnet P2P traffic.

There are several open problems in the area that may be good candidates for future research. First, there is a need to obtain more behavioral models of other types of network traffic. These new models may be click-fraud attacks, DoS attacks, data exfiltration, normal applications, etc. This growing database of behaviors is a core part of the detection performance of the CCDetector. The second open problem is the evaluation of individual behavioral models to assess its performance. We need to know how good each model is and which of them are more prone to generate errors. The third open problem is the implementation and research of an IPS (Intrusion Prevention System) that can automatically use the decisions of the CCDetector to block the traffic in real time. If this real-time blocking is combined with the detection of normal traffic, it may be possible to implement a high-level filtering system that works with rules such as *"Block botnet C&C channels but allow normal periodic applications"* or *"Block SPAM but allow normal email connections"*. Such a behavioral-based IPS may prove very useful to stop the most advanced and changing networks attacks.

9.2 Papers Published

Through the course of this thesis we published the main parts of our research in several journal and conferences. The list of published papers is:

- *An Empirical Comparison of Botnet Detection Methods*. Garcia Sebastian, Zunino Alejandro and Campo Marcelo. Computers & Security Journal. Elsevier. issn:0167-4048. doi:http://dx.doi.org/10.1016/j.cose.2014.05.011. (0) pp 100 – 123. Vol 45. 2014.

- *Survey on Network-based Botnet Detection Methods*. Garcia Sebastian, Zunino Alejandor and Campo Marcelo. doi:10.1002/sec.800. Security and Communication Networks Journal, John Wiley & Sons, Ltd. (5), pp 878–903, vol 7, 2013.

- *Botnet Behavior Detection using Network Synchronism*. Garcia Sebastian, Zunino Alejandro amd Campo Marcelo. Book chapter in 'Privacy, Intrusion Detection and Response: Technologies for Protecting Networks'. doi:10.4018/978-1-60960-836-1.ch005. Ed Kabiri, Peyman. ISBN13: 9781609608361. pp 122–144. IGI Global. 2012.

- *Identifying and Modeling Botnet C&C Behaviors*. Garcia Sebastian, Vojtěch Uhlíř. Rehak, Martin. Proceedings of the 1st International Workshop on Agents and CyberSecurity (ACySE '14 in AAMAS). doi:10.1145/2602945.2602949. ACM. 2014.

- *Detecting Botnet Traffic from a Single Host*. Garcia Sebastian, Zunino Alejandro and Campo Marcelo. Book chapter in the Handbook of Research on Emerging Developments in Data Privacy. IGI Global. Ed: Gupta, Manish. 2014. In press.

- *Botnet Behavior Detection using Network Synchronism*. Garcia Sebastian, Zunino Alejandro and Campo Marcelo. Proceedings of the Argentine Symposium on Technology. pp 1739–1750. 2010.

- *Botnets Command and Control Channels Detection using Network Behavioral Models*. Garcia Sebastian, Zunino Alejandro. Computers & Security Journal. Elsevier. Under review.

- *Botnets Behavioral Patterns in the Network. Analysis, Monitoring, Detection and Blocking*. Garcia Sebastian. Proceedings of the HackLu'14 Conference. Luxemburg. Luxemburg. 2014.

- *Identifying and Detecting Botnet Behaviors in the Network*. Garcia Sebastian. BSides Conference. Vienna. Austria. 2014.

Appendix A

A Deep Description of the Botnet Scenarios in the CCDetector Dataset

A.1 Description of the behaviors in each Scenario in the Dataset

This appendix describes in great detail the actions and behaviors of each of the thirteen botnet scenarios created in the dataset for the CCDetector method. The focus is on using our state-based behavioral model described in Subsection 6.2.3 to analyze the C&C channels and their behaviors. The goal of this appendix is to show the exact details of what is happening inside the C&C channels, which are their main characteristics and why should the CCDetector method detect them.

A.1.1 Description of Scenario 1

This scenario is part of the testing dataset and corresponds to a Neris botnet (MD5 bf08e6b02e00d2bc6dd493e93e69872f) that run for 6.15 hours in a University network. The botnet used an HTTP based C&C channel and not an IRC C&C channel as it was erroneously reported in [30]. The actions of the botnet were to communicate using several C&C channels and then to try to send SPAM, to actually send SPAM and perform click-fraud using some advertisement services. The following paragraphs describe the C&C channels in this scenario.

Label: From-Botnet-V42-TCP-CC55-Custom-Encryption This label was assigned to the 4-tuple *147.32.84.165-213.246.53.125-5296-tcp* which generated the behavioral state model *44ddi0z0i0i0i*. Most of the packets in this connection had approximately 10 bytes

and their content was probably encrypted. We are not sure about the purpose of this C&C. Due to its timeouts and non-periodic flows, this is not a representative behavioral model.

From-Botnet-V42-TCP-CC1-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-61.167.116.133-6667-tcp* which generated the behavioral state model *88yzy*. This connection is an example of a real C&C channel that sent few flows and that is not periodic. This is not a representative model for C&C connections. An example of the commands sent are:

```
POST /?c799959d9582d499959791949482d19995939782d2999790969182c699959c949c92
959c82c0999582d79995969c959d9d9482c199e79ef8f3edeae0ebf3f7f8f0e1e9f4f893ccd
dddcccad3c28ac1dcc182c399cdcacdd0a4 HTTP/1.1

HTTP/1.1 200 OK
Date: Wed, 10 Aug 2011 09:41:53 GMT
Server: Apache/2.2.8 (Fedora) DAV/2 PHP/5.2.6 mod\_ssl/2.2.8 OpenSSL/0.9.8g
X-Powered-By: PHP/5.2.6
Content-Length: 26
Connection: close
Content-Type: text/html; charset=UTF-8

CB2=212.117.171.138:65500
```

It can be seen that the connection is in fact a C&C because the IP address and port sent in the response is later used as a private SMTP server for SPAM sending (maybe proxy).

Label:From-Botnet-V42-TCP-CC53-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-31.192.109.167-80-tcp* which generated the behavioral state model *44DddddddddfddddddddddddddddddddiddddddddDDddddddDddd*. This is a good example of a not encrypted C&C were we can see the commands, and it is a good representative of the C&C connections. An example of the traffic is:

```
POST /snapbn/gate.php HTTP/1.0
Host: finalcortex.com
Keep-Alive: 300
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 56

id=SARUMAN_610d402662842e9f&version=1337&os=2600&s5=6906


HTTP/1.1 200 OK
Date: Wed, 10 Aug 2011 09:08:48 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 3
Connection: close
Content-Type: text/plain; charset=UTF-8

120
```

It can confirmed as a C&C channel because the bot is sending the name of the infected machined to the C&C server.

Label:From-Botnet-V42-TCP-CC54-Custom-Encryption This label was assigned to the 4-tuple *147.32.84.165-222.88.205.195-443-tcp* which generated the behavioral state model *66fiffiiiiiffiffffiffi*. This connection used the port 443 but did not used the protocol HTTP nor TLS. It has a good periodicity and it was custom encrypted.

Label:From-Botnet-V42-TCP-CC6-Plain-HTTP-Encrypted-Data This label was assigned to the 4-tuple *147.32.84.165-60.190.223.75-888-tcp* which generated the behavioral state model *88hhhyyhhh*. This not encrypted C&C sent custom encrypted data to the server. This is a good representative of the C&C connections. An example of its traffic is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9E
CA825FF9F6D9DFE10E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5FF432C&v=2
&t=0.7304956 HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180; Windows NT 5.1.2600)
Host: w.nucleardiscover.com:888
Connection: Keep-Alive
Pragma: no-cache


HTTP/1.1 200 OK
```

Connection: keep-alive
Date: Wed, 10 Aug 2011 10:06:49 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-Powered-By: Zend Core/2.5.0 PHP/5.2.4
Content-Length: 1536
Content-Type: text/html
1A1CA176B83E79E6769D4D10433BCA08E37F5A62D382D749DDBBF7BE70DA4D6095D9846DB1
DDC89494A3BC20D9D5957C2AED8E33F8897A2BAECB86FC47EEBAD50B4534948F23FD7EB7F5
7BF0EA1D3AE585AA27A693515A6972D8BCDF23C1A60DD954DFD7F290C71439CD43CBECE65F
C7EC1A85DEA420FC94001CDDB0E24E806E989503A884ABA917F59C49C7EC52E4569F8C2D19
CA1D0B1DC17A74E0879145433AA61194207E9E6B6A5BD9FCADDB89004B1E62F8629DF652D2
3BF79B5D421EDC2D7107F6F0C4B166DA81F783C8A388A727BFC7E8AB818FE4219E630916B7
00C67AD616F18B2AA43A68B3027CCF7D3B1414340104DDBDAF0E8F17DCAEE7254D62818769
C7ED6AB968A04D94BB9DD3B475BE1E7840E9F4C658FECA5119B51C0A263D717365B884B686
3B51F66A232857B988440DEA36421A384170EF45248C55CB1F51C464FB57E86BEFAD69E2DD
2AAA75E3CC8F0E589A8AB9DC76086B21C386FF2777ABA3F1EB1D83B672C998E02A722E65D2
83A0F1BCBC86466EFDBF29472C9C8BB1C61CE88CFA0C0C6C69FBDA64B207F0EA8BBE409778
6E873C7CE8E1F76365E07CA4215A0497621021B5902D5B48C1D1840A9043BC4EEA5DB41975
322D74B6326E66970F3B21F6A3F8D3A713788FA0DE46C4EBD5FF325955EAF19B6DCC75B329
855BBC40E17AE4885340 3EBD0F002F98B87D78D1B11BBAF068493B6AA8E8C7959300E508CF
A275918D98F26517B62D077690FB257E64BE8AEF53D01AF17D2A5EDA827391AEF6DD7D4804
42D4378D4E6BF4B510A6FA6C63CBD4B6D2FF981349556FB3D0DC631D8429C40E3750FB07C5
04E35C71C3ADB85B66F60D59885BCFAC320DB8429E977CF6C76AE9AAE65704472089DECC97
A188A90A04113B2A7758F9E417089B9C8152FD092842023D8D9BF05A90B759D73D3D11F98E
C887804AE124B7F40A62C8864D45480FDCE4AE22C54CB6E322B8FB04FE5A25CC1E726C73CD
0F6F33B647645034E39CC734406C07F4DBC35BBB94567C563D57E887ED3998488E6EC235D2
1EBF7FE101DA1F619E2CB798B9991F1A3A5AD57446DEA2D0B476EDC2

Label:From-Botnet-V42-TCP-CC16-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-173.192.170.88-80-tcp* which generated the behavioral state model

96fFIFIIFfiiiiIFFiFififfIfifIffIFfIfifIiiififIFfIfifIFifffiifIiIiIFiiFIifi
iifIFfifiFIfHIIiififIFIIfigifIFIFII0zzIzwz0wzz0zzzziiIIiIFiifiiiiifIIFFFFf
ffifiiIFzzzzzwiIfizzzIFwzhhI0zw0zwwzwwfifIfIiiIIffiIiIiIfFiiifiiffffiiifii
iiffiififfiiiiffifiiffifiIIfifiiiffiIIizwIiIIiF

This connection used a non-encrypted HTTP protocol to send probably encrypted data. This is a good representative of the C&C connections. An example of its packets is:

```
GET /g1.php?u=hhhh&v=6&m=04FCFD6F-083E-4377-A1F3-FEF727AEF499 HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; waoc)
Host: www.lddwj.com
Cache-Control: no-cache


HTTP/1.1 200 OK
Date: Wed, 10 Aug 2011 09:10:08 GMT
Server: Apache/2.2.19 (Unix) mod_ssl/2.2.19 OpenSSL/0.9.8e-fips-rhel5
mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635
X-Powered-By: PHP/5.2.17
Transfer-Encoding: chunked
Content-Type: text/html

3
...

JBFXDEHYEXJM
-----
EUHYBOIQGJEOILGWELFR
-----
JHGKBNBIIVENGPGCGFCTFJFIFAJDEUIIHI
JHGKBNBIIVENGSBUBWCXFUCQBEHJEFAUEUDBHEGFDSDABNHHEXBMAQHKIBASBAAEAVIJ
JHHBEXJDBRGTISJCBWJKDSJKBYGJGTDVCCBVIQIYGBEHFXEBGZBWGGDOBAIO
JHGKBNBIIVENGMJJJQAVFKAKETAGGDHQ
-----
DUHNHLCSCEGRDPJQAYDJBB
-----
JHHBEXJDBRHIIHGJCDCQHUHHHTBWAEJPAYCXJSDKCYGNDUDLDJGGIYHOGF
JGFOFZCFCLHJFRGAIGIADKEUISIZBODMEHITAWIBBPCSDOCAETBDIHJVGTEBIXINHBGWDBGW
JAEJBSHSCLIGFUCTCWFFGJCZJCJS
JHHBEXJDBRGHFNACGWJRFXBRDKAQAXBYGHAWBLCSEDBGJMJVJLDHAI
IXHVCNDEHQIFCFDQEXEBFRBIFHDBBMHGGRCGJSGCIZIUHZFZAXEHGDDZAGIVGXFWJSAPCJFY
AACMCPJAAQBZDIGFFIHDHNIPAGHSDVENHOBMGQCGEWIWAFCYADCWAKFRCXGFCACNFCCS
```

```
-----
DUHNHLCSCEGRDPJQAYDJBB
-----
JHHBEXJDBRHIIHGJCDCQHUHHHTBWAEJPAYCXJSDKCUJSIXAGAZBGABCBEABNJSHYHQGACFHF
BCBDFADRFHEJJPCLIGBNIPHFGLEDEOIGEVHIHJEZHWAIJIFDDDEMDVEAGOENEVDUHFAIAPJV
HOBVHCDHBVHRDTACBIGOHTBHCTADILHOASBYCRDSHBCRDLEZGIFOJMHXHKCOICAPESGGJQJS
AWHHFFJGHPEYFPFGFFAHBTAPBXDCCUGVAGAOBQDKCHBCFGGUCEHODJGYECHTBYAHHMEGGKDJ
IXGZGGJIBMFAAYBFAN
JGFOFZCFCLHJFRGAIGIADKEUISIZBODMEHITAWIGBNJMEKCYARDMIHGTJCCOBPFU
JAEJBSHSCLIGFUCTAFDFDMFAJFHUHTJBGNHT
JHHBEXJDBRGHFNACGWJRFXBRDKADFPGRAPIHFZGWDYFMFU
IXHVCNDEHQIFCFDQEXEBFRDQDQGRGNGSJBHNBEGQGDHMCLDXHZGSHXBSCACWIZDGATGQBKES
EUIGIVFOHWBSESDYISATIEEDIBFGIZEPBIJJAPGADQAJDFDICHHBHQJEFIHUAEGUIAFHIVEP
HLILDRERCWHSEKCEHKDR
```

A.1.2 Description of Scenario 2

This scenario is part of the testing dataset and corresponds to a Neris botnet (MD5 bf08e6b02e00d2bc6dd493e93e69872f) that run for 4.21 hours. This is the same botnet as Scenario 1. The botnet used an HTTP-based C&C channel and not an IRC C&C channel as it was erroneously repored in [30]. The actions of the botnet were to communicate using several C&C channels and then to try to send SPAM, to actually send SPAM and perform click-fraud using some advertisement services. The following paragraphs describe the C&C channels in this scenario.

Label:From-Botnet-V43-TCP-CC1-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-217.34.4.225-6667-tcp* which generated the behavioral state model *11r*. This is an example of a real C&C that sent few flows. This is not a representative model for C&C connections. An example of its packets is:

```
POST /?540a03050211470a00060f0511420a01000411410a040305021155 0a06020000
01040411530a0611440a060e020007040311520a740d6b607e79737860646b63727a676
b475a565c06065c4619524f5211500a5e595e4337 HTTP/1.1
Accept-Language: en-us
CB2: 1
User-Agent: Mozilla
Host: 217.34.4.225
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Thu, 11 Aug 2011 09:50:15 GMT
Server: Apache/2.2.8 (Fedora) DAV/2 PHP/5.2.6 mod_ssl/2.2.8
OpenSSL/0.9.8g
X-Powered-By: PHP/5.2.6
Content-Length: 26
Connection: close
Content-Type: text/html; charset=UTF-8

CB2=212.117.171.138:65500
```

The IP address that was sent in the response was later used for SPAM sending.

Label:From-Botnet-V43-TCP-CC6-Plain-HTTP-Encrypted-Data This label was assigned to the 4-tuple *147.32.84.165-60.190.223.75-888-tcp,* which generated the behavioral state model *11raArrrAA*. This connection used the HTTP protocol to send probably encrypted data. This is a good but short representative model of C&C connections. An example of its packets is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9ECA825
FF9F6D9DFE0E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5FF432C
&v=2&t=0.2840387 HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180; Windows NT 5.1.2600)
Host: w.nucleardiscover.com:888
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Connection: keep-alive
Date: Thu, 11 Aug 2011 09:38:01 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-Powered-By: Zend Core/2.5.0 PHP/5.2.4
Content-Length: 1536
Content-Type: text/html

1412CE196AEC20BF28C3C69BACD49250118D90A8E7B68D13ADCB682174DE6944460AE60
```

```
F95F9BCE088BF53CF1D11876E8542A31E245533627316740E9A33F996044A0EAEFD51E4
670745B63DF70016C93C1345C430F27043EF45741719FB51FA6BE6B4BCFC9E8B58B6420
68E131935ADB741B6EDF67287EF3A266805208C6F81EBE650FBD6F9F14FBED70E803B85
0FBDDCCF0E3AD90E5B4D972CCE5A5B4DD4D267FB0184D48A41B483B2EACF483EA62F144
1DB41D22D0AAE947D432F160927E555092EDF665233E4C59EA3D73F
(...)
```

Label:From-Botnet-V43-TCP-CC66-HTTP-Custom-Encryption This label was assigned to 4-tuple *147.32.84.165-184.82.148.43-80-tcp* which generated the behavioral state model

```
11aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaararrrArrraraarraaaraAraraArararaaat
rraArarrrrrraAarrCrrrrrrrrrrrrarrrrrrrArrArrrrrrrrrAarrrrrraArrrrrArrrr
trrrrrrrrrcrrArArrrrrrrrArrrtrrraaarrrrar
```

This connection used the HTTP protocol to send data. This is a good representative model for C&C connections. An example of its packets its:

```
POST /getjson HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
Host: podviajar.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Content-Length: 417
Connection: close
Cache-Control: no-cache

data=e/e6r8VdRlJajwxXJgogFVa9gSXExozmaI7cOYwrAvLDx2T2KL4nXdZonsue0ZZR7
3lEXxz8TzdT6x30edcsdwMJOdAF/VjVsWHF0Ey7zDJW9/s9JErJ0pfr82QY6b8HCgSuNaU
qig4oVzoegralxi+4RGu6GXngv/2Z4yLfilVPRaJU0eP/NkoNLov8k5s5mypmXMF5apCqD
xNCgYneZ1/mCv99lA0iEQ+oWXiJbP4U/mJShBEMRT+yUrYV+hYDQsj8kIgeLxe0VANXrqf
B5R+x3RibNlu5I/wSRF5lSctQ4uSrq1VRtlbVOLT+WTcp9EEI1mgAayBbKU0WYqSGq09ph
QmcVhGKO01MloGt/3fcbQZL7jGcDx7EwR37wG7rFfhpwaOyPEQAdY+Jpnrurt/O7K0=


HTTP/1.1 302 Moved Temporarily
Server: nginx/1.0.0
Date: Thu, 11 Aug 2011 08:40:18 GMT
Content-Type: text/html
```

```
Content-Length: 160
Connection: close
Expires: Thu Aug 11 07:40:18 2011
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: session=Zuqq4stGXQVYyBQEbFh2Qlqojy6Yl4fhcNWJaod/G6KVmg==
Set-Cookie: id=YPGw9w==
Set-Cookie: counter=8
Set-Cookie: stime=ff2s+4Ed
Location: http://creditskin.com/getjson?4pq7f61CqbOfvrJVH6taYoAq3rBYi
P8ofCuBkRZuxw==

<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.0.0</center>
</body>
</html>
```

Label:From-Botnet-V43-TCP-CC69-Custom-Encryption This label was assigned to the 4-tuple *147.32.84.165-184.82.147.251-80-tcp*, which generated the behavioral state model:

```
31aarraarArArrAtrrAArrarrAAAACrArrrAAAAarrraraaArarrrArraarrrarrrarrr
CaArraararAraaarArraaAttsrrraarrArArArrrrAaarrrarrAArrrArArrrArrarAaa
AaarArArrArrrrrrrarrraAAAaAArrArraAaaaAArrrrrrrrrrrrArrrrrraA
```

This connection was a good example of a C&C channel with a long chain of states that used the HTTP protocol to send probably encrypted data. The responses include a Location header that changed over time. This is a good representative model for C&C connections. An example of its packets is:

```
POST /getjson HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
Host: slapcube.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

```
Content-Length: 425
Connection: close
Cache-Control: no-cache

data=e/e6r8VdRlJajwxXJgogFVa9gSXExozmaI7cOYwrAvLDx2T2KL4nXdZonsue0ZZR7
3lEXxz8zdT6x30edcsdwMJOdAF/VjVsWHF0Ey7zDJW9/s9JErJ0pfr82QY6b8HCgSuNaUq
ig4oVzoegrelySD3VSq0TCbovu/BtGCNh1dfVLBEgLu2dV5Dcdqog5Y2nCQzC5JnMsbhSg
hJlZaPMgi1C6wz11VnCQe0QyTZLq5d5joRxU5WDXTkBqEZ9w1ZTID3m49NKUm5Xh1G5KfM
4Q+gzhHHPVLhN+hRDEhuFt9Eos82pfq1rXIOicAmNwuAUKljFeOnQaKE4eKeWkWRAwASSK
3njKQQU3N7hrAD7V0IRt3OHYRmyxRrsmCqFArwnx+DuITdFI56FrGrmrNrF8aamQy/N2UJA


HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Thu, 11 Aug 2011 08:55:15 GMT
Content-Type: text/html
Content-Length: 154
Connection: close
Expires: Thu Aug 11 07:55:15 2011
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: session=Zuqq4stGXQVYyBQEbFh2Qlqojy6Yl4fhcNWJaod/G6KVmg==
Set-Cookie: id=YPGw9w==
Set-Cookie: counter=8
Set-Cookie: stime=ff2s+4Ed
Location: http://splashpops.com/getjson?4pq7f6Mkx6TQp
/9ZGqhDf4E2yLxCl/grb2qDm0QQ3tw=

<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Label:From-Botnet-V43-TCP-CC70-Custom-Encryption This label was assigned to the 4-tuple *147.32.84.165-184.82.155.107-80-tcp*, which generated the behavioral state

model *11arrAtaarrrrrrrrArrrrAAarrrArAtrrcrrrarcrrrraaCrrrrrAAArrrrrarrrAAarrrrrrrrrrrrrrrrrrr*. This is a similar behavior than the previous one, but longer. The C&C channel used the HTTP protocol with probably encrypted data. The lack of a strong periodicity for long periods of time may suggest that it is not part of the main C&C channen, but maybe a sub-channel. This is a not so good representative of the C&C connections. An example of its traffic is:

```
POST /getjson HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
Host: podwine.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Content-Length: 325
Connection: close
Cache-Control: no-cache

data=e/e6r8VdRlJajwxXJgogFVa9gSXExozmaI7cOYwrAvLDx2T2KL4nXdZonsue0ZZR73l
EXxz8TzdT6x30edcsdwMJOdAF/VjVsWHF0Ey7zDJW9/s9JErJ0pfr82QY6b8HCgSuNaUqig4
oV3BBwK2t2z73RGeyUDjgsuHFpCLOikRPSL9SjuP1IB8bKpjtmJ0ig3VmVcFmboKsDRhYnYu
CZ1T0VLh4hR4yDAmpCnacPqJQ/WAdmRcNTiTgF7EV6BcIHsj0i4xfK0myX15WseDa8Q+i2xr
DPUGpcOxQDGx1XIoXv6rtpikP4A2JbuflVygg

HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Thu, 11 Aug 2011 08:38:10 GMT
Content-Type: text/html
Content-Length: 154
Connection: close
Expires: Thu Aug 11 07:38:10 2011
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: session=Zuqq4stGXQVYyBQEbFh2Qlqojy6Yl4fhcNWJaod/G6KVmg==
Set-Cookie: id=YPGw9w==
Set-Cookie: counter=8
Set-Cookie: stime=ff2s+4Ed
Location: http://scubavests.com/getjson?4pq7f6RCqbOfvrJVH6taYoAq
3rBYiP8ofCuBkRZuxw==

<html>
```

```
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Label:From-Botnet-V43-TCP-CC56-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-31.192.109.161-80-tcp*, which generated the behavioral state model *11aaaarrrrrAAarArrraAAraaarrrArrrrrarraAraarraaaa*. This is a good example of a periodic C&C channel with some non-periodic flows using the HTTP protocol and non-encrypted data. The IP addresses seen in the response are later used by the bot. An example of its packets is:

```
POST /fakedream/index.php HTTP/1.0
Host: xzrw1q.com
Keep-Alive: 300
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US)
Content-Type: application/x-www-form-urlencoded
Content-Length: 17

k=880313430709865

HTTP/1.1 200 OK
Date: Thu, 11 Aug 2011 08:39:17 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 146
Connection: close
Content-Type: text/html; charset=UTF-8

01|500|60http://193.23.181.44:80
http://193.23.181.44:179
http://174.37.196.55
http://72.20.15.61
http://174.128.246.102
http://67.19.72.206
```

```
POST /fakedream/index.php HTTP/1.0
Host: xzrw1q.com
Keep-Alive: 300
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US)
Content-Type: application/x-www-form-urlencoded
Content-Length: 17

k=880313430709865



HTTP/1.1 200 OK
Date: Thu, 11 Aug 2011 10:12:30 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 245
Connection: close
Content-Type: text/html; charset=UTF-8

14|500|60http://www.guljan.org/javascript/send_form.js?a=()&b=()!1425
http://www.guljan.org/ajax_send_resume.php?c=()()()!1002
http://www.guljan.org/ajax_send_save.php?d=()()()!1506
http://www.guljan.org/ajax_sent_subscribe.php?a=()()()!1604
```

Label:From-Botnet-V43-TCP-CC16-HTTP-Not-Encrypte This label was assigned to the 4-tuple *147.32.84.165-173.192.170.88-80-tcp*, which generated the behavioral state model

```
33tttstttrttswttttttttttctttfctwttttttttctttttttttttcttcttttstttttt
ttstttttrwttttttttttrrtt0tsttt0ttstrttttttttttttttttttttctttttttttttt
tttrttt
```

This is a C&C connection using the HTTP protocol and probably encrypted data. However it is non-periodic. It is still a good representative of C&C behaviors. An example of its packets is:

```
GET /g1.php?u=hhhh&v=6&m=9C1F2D00-462D-4AE0-B00D-4A55CD7002D3 HTTP/1.1
```

```
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; waoc)
Host: www.lddwj.com
Cache-Control: no-cache

TTP/1.1 200 OK
Date: Thu, 11 Aug 2011 08:41:16 GMT
Server: Apache/2.2.19 (Unix) mod_ssl/2.2.19 OpenSSL/0.9.8e-fips-rhel5
mod_auth_passthrough/2 mod_bwlimited/1.4 FrontPage/5.0.2.2635
X-Powered-By: PHP/5.2.17
Transfer-Encoding: chunked
Content-Type: text/html

3
...
JBFXDEHYEXJM
-----
EUHYBOIQGJEOILGWELFR
-----
JHGKBNBIIVENGPGCGFCTFJFIFAJDEUIIHI
JHGKBNBIIVENGSBUBWCXFUCQBEHJEFAUEUDBHEGFDSDABNHHEXBMAQHKIBASBAAEAVIJ
JHHBEXJDBRGTISJCBWJKDSJKBYGJGTDVCCBVIQIYGBEHFXEBGZBWGGDOBAI
JHGKBNBIIVENGMJJJQAVFKAKETAGGDHQ
-----
DUHNHLCSCEGRDPJQAYDJBB
-----
JHHBEXJDBRHIIHGJCDCQHUHHHTBWAEJPAYCXJSDKCYGNDUDLDJGGIYHOGF
JGFOFZCFCLHJFRGAIGIADKEUISIZBODMEHITAWIBBPCSDOCAETBDIHJVGTEBIXINHBGWDBGW
JAEJBSHSCLIGFUCTCWFFGJCZJCJS
JHHBEXJDBRGHFNACGWJRFXBRDKAQAXBYGHAWBLCSEDBGJMJVJLDHAI
IXHVCNDEHQIFCFDQEXEBFRBIFHDBBMHGGRCGJSGCIZIUHZFZAXEHGDDZAGIVGXFWJSAPCJFY
AACMCPJAAQBZDIGFFIHDHNIPAGHSDVENHOBMGQCGEWIWAFCYADCWAKFRCXGFCACNFCCS
-----
DUHNHLCSCEGRDPJQAYDJBB
-----
JHHBEXJDBRHIIHGJCDCQHUHHHTBWAEJPAYCXJSDKCUJSIXAGAZBGABCBEABNJSHYHQGACFHF
BCBDFADRFHEJJPCLIGBNIPHFGLEDEOIGEVHIHJEZHWAIJIFDDDEMDVEAGOENEVDUHFAIAPJV
HOBVHCDHBVHRDTACBIGOHTBHCTADILHOASBYCRDSHBCRDLEZGIFOJMHXHKCOICAPESGGJQJS
```

```
AWHHFFJGHPEYFPFGFFAHBTAPBXDCCUGVAGAOBQDKCHBCFGGUCEHODJGYECHTBYAHHMEGGKDJ
IXGZGGJIBMFAAYBFAN
JGFOFZCFCLHJFRGAIGIADKEUISIZBODMEHITAWIGBNJMEKCYARDMIHGTJCCOBPFU
JAEJBSHSCLIGFUCTAFDFDMFAJFHUHTJBGNHT
JHHBEXJDBRGHFNACGWJRFXBRDKADFPGRAPIHFZGWDYFMFU
IXHVCNDEHQIFCFDQEXEBFRDQDQGRGNGSJBHNBEGQGDHMCLDXHZGSHXBSCACWIZDGATGQBKES
EUIGIVFOHWBSESDYISATIEEDIBFGIZEPBIJJAPGADQAJDFDICHHBHQJEFIHUAEGUIAFHIVEP
HLILDRERCWHSEKCEHKDR
-----
```

A.1.3 Description of Scenario 3

This scenario is part of the training dataset and corresponds to an Rbot type of botnet (MD5 2467b3c8b259cecd6ce2d5c31009df10) that run for 66.85 hours. The binary of this botnet was recompiled by our team in order to control it ourselves. The idea is to force the execution of some attacks on a controlled machine using the same tools as the attackers. The main actions of the bot were: to connect a single bot to the IRC channel, to download information from the bot to the botmaster (system information, screen shots, windows commands, process list), to port scan ports 80/tcp and 22/tcp in a remote network and to reboot the bot. The following are the C&C channels in this scenario.

Label:From-Botnet-V1-TCP-CC107-IRC-Not-Encrypted This label was assigned to the 4-tuple *38.229.70.20-147.32.84.165-1027-tcp*, which generated the behavioral model state

```
360t0t0c0c0c0f0f0t0t0c0c0c0c0c0c0f0t0t0c0c0c0c0c0c0c0c0c0w0F0w0t0t0f0c0c0c
0c0c0f0c0c0c0c0c0c0c0c0c0c0c0i0w0w0w0w0w0w0w0w0w0w
```

The main characteristic of this chain of states is that the timeout is reached after each flow. This is because the IRC protocol uses only one flow (only one source port) for its communication. Therefore, the only separation of flows is when the timeout is reached. Despite this, the flows are periodic most of the time because the IRC protocol has a heartbeat mechanism for keeping the connection alive. It is a good representative of an IRC C&C. An example of the content of the IRC channel is:

```
NICK Pepe061664
USER uyzzmmgi 0 0 :Pepe061664
```

```
MODE Pepe061664 -x
JOIN #zarasa48
PRIVMSG #zarasa48 :.sysinfo
PRIVMSG #zarasa48 :System [CPU]: 3175MHz. [RAM]: 1,048,048KB total,
1,048,048KB free. [Disk]: 52,420,060KB total, 48,810,972KB free.
[OS]: Windows XP (Service Pack 2) (5.1, Build 2600). [Sysdir]:
C:\WINDOWS\system32. [Hostname]: saruman (147.32.84.165).
[Current User]: Ctu. [Date]: 12:Aug:2011. [Time]: 16:14:23.
[Uptime]: 0d 0h 4m.
PING :card.freenode.net
PONG :card.freenode.net
```

It should be noted that the 4-tuple has as source IP address the IRC server instead of the bot because some packets were lost at the beginning of the capture due to some technical issues.

A.1.4 Description of Scenario 4

This scenario is part of the training dataset and corresponds to an Rbot type of botnet (MD5 2467b3c8b259cecd6ce2d5c31009df10) that run for 4.21 hours. The binary of this botnet was recompiled by our team in order to control it ourselves. The idea is to force the execution of some attacks on a controlled machine using the same tools as the attackers. The main actions of the bot were: to connect a single bot to the IRC channel, to download information from the bot to the botmaster (system information), to perform a DoS attack using SYN packets to one remote machine under our control, to perform a DoS attack using ACK packets to the same target and to perform a DoS attack using ICMP packets to the same target. In this scenario, the IRC communication lasted for 26 minutes, and so the IRC flow did not timeout. Then, there was only one C&C flow and there was not enough information to create our behavioral model. This is an example of how small attacks can not be captured by our model.

A.1.5 Description of Scenario 5

This scenario is part of the training dataset and corresponds to a Virut botnet (MD5 85f9a5247afbe51e64794193f1dd72eb) that run for 11.63 hours. The main actions of this botnet include: download some binary executables files, sent SPAM and intercept hotmail connections using the bot as a web-proxy (port 65500). The following paragraph*s are the C&C channels in this scenario.

Label:From-Botnet-V46-TCP-CC12-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-91.220.0.52-80-tcp*, which generated the behavioral state model *2rBACrtrtstr*. This connection uses the HTTP protocol with probably encrypted data on port 80/TCP. The flows are mostly not-periodic, so this is an example of a C&C channel that does not need the periodicity. It is however, a good example of a C&C. An example of its packets is:

```
POST /ajax.php HTTP/1.1
Host: 897234kjdsf4523234.com
Content-Length: 217
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Connection: close

9f94ee2e06c3cb29=kFtCLYAVZaEnL%2BbLz%2BhTEA3YUws86YUsh1BGuDrowHWT7k4wq
slD5NDHFhL1OP3t%0D%0Adrm2EA2ew25MHJlav30eMYtxSlPfPoGN5YWUP%2FRyxaf6eSQ
BSlNcUhDmNItD%2BFNA%0D%0AhSWfUgTJ7a8YUj8yCRzwMBhnw7Xx5RcL2ZysVg909jA
%3D%0D%0A

HTTP/1.1 200 OK
Server: nginx/1.1.0
Date: Mon, 15 Aug 2011 14:55:42 GMT
Content-Type: text/html
Connection: close
Content-Length: 25

Vohoo28z4XLVyj/LtqcoLw==
```

Label:From-Botnet-V46-TCP-CC5-Plain-HTTP-Encrypted-Data This label was assigned to the 4-tuple *147.32.84.165-94.63.150.63-80-tcp*, which generated the behavioral state model *41rrrAr*. This connection used the HTTP protocol with probably encrypted data. The encryption schema seems to be different that the previous 4-tuple. The flows are mostly not-periodic, so this is an example of a C&C channel that does not need the periodicity. It is however, a good example of a C&C. An example of its packets is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9ECA825F
F9F6D9DFE10E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5F44337&v=2&t=0.659115
HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180; Windows NT 5.1.2600)
```

```
Host: mewgost.com
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Date: Mon, 15 Aug 2011 17:00:09 GMT
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch
X-Powered-By: PHP/5.2.6-1+lenny9
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

1B1DE14A75F640E15BBC351B5B2C7DBD9B02E7DC2C0A1388187A6726A0050026205EC72D1
07A0658BA849B045C4E0AEDC500BD55D7A4391D4524730705AF600F542A7BDBA10DB033B0
F2FC772DDA4B94E3CC59D85E9CFCCFC16BFE9DFC1EE150C6480C31E8B08156A85844D68B7
56E3F02EE6203A7244878D6B4ADC2A9032ADB282A4AECDAEBD9184771084624D12BE8626D
(...)
```

A.1.6 Description of Scenario 6

This scenario is part of the testing dataset and corresponds to a Menti (maybe DonBot) botnet variant (MD5 66b8864b660eae1bfb9750b1b3e9b449) that run for 2.18 hours. The main actions of the botnet were to create and use a custom protocol for an unencrypted C&C channel and to scan for ports 25/TCP searching for SMTP servers (it did not sent SPAM). The following is the C&C channel of the botnet:

Label:From-Botnet-V47-TCP-CC73 This label was assigned to the 4-tuple *147.32.84.165-91.212.135.158-5678-tcp*, which generated the behavioral state model

```
71rrrrrrraaarrrrrwrrrraarrrrrwarrrarrrrrrrrarrrraarrrrrrrrraararrrrrrr
rarrarrrrrrrrrrrrrrrrrrrrrrrarraarrrrrrrrrabrrrrarraarrrrrrrarrrrrarara
rrraararrrrraaarrrrrrrrarrsrrrarraaaararrrraaaaararaaaa
```

An example of its packets is:

```
Hash: 66b8864b660eae1bfb9750b1b3e9b449
ID: svchosta
Session:
```

```
Domain: NA
RBL: 0
Sent: 0
Failed: 0
Catchall: 0

CHUNK

Session: f6c18a266f5494c8df66e4f05b0fe79f
IP: 147.32.84.165
Keep-Alive: 1
RBL: bl.spamcop.net
Max-To: 6
Max-Threads: 60
ProxyLock: 0
BlockCatchalls: 0
Clear Buffers
Macro: 26
DIGIT
0
1
2
3
4
5
6
7
8
9
Macro: 385
FMNAME
james
john
robert
michael
william
david
richard
charles
```

```
joseph
(...)
Macro: 4167
FNAME
abbey
abby
abigail
adrian
adriana
(...)
Macro: 12064
FROMDOMAIN
100megas.com
101mail.net
1colony.com
1stopkorea.com
1usamall.com
2trom.com
321webmaster.com
(...)
Macro: 26100
HOTMAILS
ffonymdilell@hotmail.com
ggachooredo@hotmail.com
krurombech@hotmail.com
rchepamboossa@hotmail.com
rcepidashalli@hotmail.com
mauvescorrela@hotmail.com
asemyratarllev@hotmail.com
(...)
Macro: 175
SUBJECT
hi sweetie
hey honey..
it‘s me again
remember me?
(...)
```

A.1.7 Description of Scenario 7

This scenario is part of the training dataset and corresponds to a Sogou botnet (MD5 8a71965cba1d3596745f63e3d8a5ac3f) that run for 0.38 hours. The main actions were to connect to a non encrypted HTTP C&C channel, to access google search services, to download compressed files, and to download binary files. There was no attack. The following paragraph are the C&C channels in the scenario.

Label:From-Botnet-V48-TCP-CC77-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-61.135.188.210-80-tcp*, which generated the behavioral state model *23Aattr*. This channel used the HTTP protocol to send non-encrypted data. Some of the connections were answered with a 404 Not found page, which may mean that part of the C&C was down. This is not a good representative of a C&C behavior. An example of its packets is:

```
GET /ie.png?15DF37%7nEe7r5sC6A20%982184F.005.D43.%1hD632%%v66321%24DE39%3r
BFA34%Bf1iFg3.Ah2cCr3a3es HTTP/1.1
Host: ping.ie.sogou.com
Cache-Control: no-cache

HTTP/1.1 404 Not Found
Server: nginx/0.7.62
Date: Tue, 16 Aug 2011 11:56:28 GMT
Content-Type: image/png
Content-Length: 0
Connection: close


GET /version_sogoubrowser.txt?r=2136&h=50F8290AC57E77F533C2A3F1B4AB39E4
&v=1.4.0.418&s=1&sf=1&IE=6.0.2900.2180 HTTP/1.1
User-Agent: SOGOU_UPDATER
Host: config.ie.sogou.com
Cache-Control: no-cache
Cookie: YYID=50F8290AC57E77F533C2A3F1B4AB39E4

HTTP/1.1 200 OK
Server: nginx/0.7.62
Date: Tue, 16 Aug 2011 11:56:49 GMT
```

```
Content-Type: text/plain;charset=gbk
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/5.1.6

123

[product0]
pid=3
name=.........
size=15.81M
officalurl=http://ie.sogou.com
html=http://10.10.66.55/ie/pinyin.html
icon=http://10.10.66.55/ie/pinyin.ico
version=2.2.0.2070
date=2011-05-09
url=http://files.sogou.com/sogou_explorer_upgrade_2.2.0.2070.exe
md5=323941ef2f6de7aa2e0432be2ed4b156
```

Label:From-Botnet-V48-TCP-CC62-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-123.126.51.33-80-tcp*, which generated the behavioral state model *99ciiiiizziiittcccc*. This channel used the HTTP protocol without any encryption and it is a good representative of a periodic C&C channels.

```
GET /?t=1&s=2&m=50F8290AC57E77F533C2A3F1B4AB39E4 HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
SE 1.X)
Host: 123.ie.sogou.com
Connection: Keep-Alive


HTTP/1.1 200 OK
Server: nginx/1.0.5
Date: Tue, 16 Aug 2011 11:56:37 GMT
Content-Type: text/html; charset=gb2312
```

```
Connection: keep-alive
Last-Modified: Mon, 15 Aug 2011 09:56:46 GMT
ETag: "4d7cd3-6e64-4aa8847ef9380"
Cache-Control: max-age=300
Expires: Tue, 16 Aug 2011 11:56:47 GMT
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 9000
Accept-Ranges: bytes
(...)


GET /features.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
SE 1.X)
Host: ie.sogou.com
Connection: Keep-Alive



HTTP/1.1 200 OK
Server: nginx/1.0.5
Date: Tue, 16 Aug 2011 11:56:37 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 11368
Last-Modified: Mon, 28 Mar 2011 11:55:11 GMT
Cache-Control: no-cache
Accept-Ranges: bytes

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>.................</title>
```

(...)

A.1.8 Description of Scenario 8

This scenario is part of the testing dataset and corresponds to a Murlo botnet (MD5 268663702c32435db6fe4b24f962796b) that run for 19.5 hours. This is a large capture that is full of different behaviors. The main actions of the botnet were: to download binary files, to use a non-encrypted C&C channel, scan the local network looking for port 135/TCP and have some P2P-like UDP connections. However, more interesting are the dynamics of the connections, how it can be seen that after each NetBios scan the bot is sending the results back the the botnet, how different C&C channels start or stop the botnets actions. The following paragraphs are the C&C channels.

Label:From-Botnet-V49-TCP-CC74-HTTP-Custom-Port-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-222.189.228.111-3389-tcp*, which generated the behavioral state model:

```
33Ccccccccccccc CCcCttttcccttcttcccccccccccccccccccCCcttcctcccccccccc
ccccccccccccccccccccccttttcccccccccccccccccccccccccccccctccctcccccc
cccccccccccccccccccccCCcctCtttcccccccccccccccccccccccccccccctctttccc
CtCtttCccc0rtCttccCcCccccccccccccccccccccccccccCtcctcccccccccccccccccc
cccccccccccccctttctcccccccccctcccccccccccccccccccttcctcccCCccCCCCctt
cCCCcCccccccccccccctcctccCCccCCccccccccccccCCcCCCccCCcCCttttCCccccccttc
cttcCCcCCcCcCCCccCccCCtcCtCCccCcCcttcccccccccccccccccccCcCtCctcccctt Cccc
CccCCccctcccttcccCCcccCtcctttcccCcccCcCCcCCCcCCccCccccccCCCttttcccCttt
tttcccCcCccccCCcCcCcttccctcctccccccCCccccctttcccCccCccCcCcCCccttcctCcccc
ccccccccCCcccCCCCccccCCcCCttccttcccCCcCCCCcCCccccccccccccccttCctttttcc
CCcccCCcccCcccCtttCccttcccttcctctttccCCcCcCcttCccctttcccccccccccccccttttc
ccccccccccccccccccccccccccccccccccttttcccccccccccccccccccccccccccct
tttcccccccccccccccccccccccccccccccccccttttcccccccccccccccccccccccccc
ccccttttcccccccccccccccccccccccccccccccccttcttcccccccccccccccccccccc
cccccccttttcccccccccccccccccccccccca
```

This connection is perhaps the best representation of a full functional, not sinkholed, working and active C&C channel of our dataset. The port used is the default RDP protocol port, but the traffic is not RDP, acting as a disguise for most filters. An example of its packets is:

```
GET /tool/train/q.txt
User-Agent: VBTagEdit\right
Host: zxc.78rr.cn:3389

HTTP/1.1 200 OK
Content-Length: 96
Content-Type: text/plain
Last-Modified: Sat, 06 Aug 2011 19:52:28 GMT
Accept-Ranges: bytes
ETag: "fa6def5e7254cc1:610"
Server: Microsoft-IIS/6.0
Date: Sat, 13 Aug 2011 23:28:05 GMT

hbojs.kc18.cn
hboqq.kc18.cn
hboqqsy.kc18.cn
www.pypiao.com
zxc.78rr.cn/tool/train/update.txt


GET /tool/train/host.txt HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: zxc.78rr.cn:3389
Connection: Keep-Alive


HTTP/1.1 200 OK
Content-Length: 3043
Content-Type: text/plain
Last-Modified: Mon, 08 Aug 2011 06:58:14 GMT
Accept-Ranges: bytes
ETag: "6810e68a9855cc1:610"
Server: Microsoft-IIS/6.0
Date: Sat, 13 Aug 2011 23:28:07 GMT

this is host
222.189.228.112  minigame.qq.com
```

```
222.189.228.112  music.qq.com
222.189.228.112  show.qq.com
222.189.228.112  qqgame.qq.com
222.189.228.112  www.qq.com
222.189.228.112  minisite2009.qq.com
222.189.228.112  qqgamecdnimg.qq.com
222.189.228.112  minigameimg.qq.com
222.189.228.111  10.qq.com
(...)
127.0.0.1  3304.6600.org
127.0.0.1  www.3320.net
127.0.0.1  baidu.souyh.com
127.0.0.1  config.ie.sogou.com
127.0.0.1  www.tonlions.com
127.0.0.1  files2.sogou.com
127.0.0.1  download.pplive.com
(...)
```

Most of the periodic connections (*c* and *C* letters) corresponds to this type of requests:

```
GET /tool/train/c.txt HTTP/1.1
User-Agent: VBTagEdit
Host: zxc.78rr.cn:3389

TTP/1.1 200 OK
Content-Length: 177
Content-Type: text/plain
Last-Modified: Sat, 09 Jul 2011 13:39:06 GMT
Accept-Ranges: bytes
ETag: "ee90d933d3ecc1:610"
Server: Microsoft-IIS/6.0
Date: Sun, 14 Aug 2011 04:52:01 GMT

........:....QQ..........|
    ............................
......!............1806........
............,...................
..............!
```

```
|http://zhu.kc18.cn:3389/128.htm
```

Label:From-Botnet-V49-TCP-CC75-HTTP-Custom-Port-Not-Encrypted-Non-Periodic
This label was assigned to the 4-tuple *147.32.84.165-222.73.45.135-81-tcp*, which generated the behavioral state model *31sw0r0A0s0r0A0r0s0b0s0r0r0s*. This connection had a long periodicity of one hour, however since there were small time differences of several seconds between the flows, the letters correspond to non-periodic flows. An example of its packets is.

```
GET /rc/xms/gx2.txt HTTP/1.1
User-Agent: API-Guide test program
Host: www.caifu5678.cn:81
Cache-Control: no-cache

HTTP/1.1 200 OK
Content-Length: 1
Content-Type: text/plain
Last-Modified: Wed, 17 Jun 2009 15:37:39 GMT
Accept-Ranges: bytes
ETag: "9cf1ab8b61efc91:3e0f"
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Tue, 16 Aug 2011 12:38:19 GMT

0

GET /rc/zj/wwan.jpg HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: www.caifu5678.cn:81
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Length: 12010
Content-Type: image/jpeg
Last-Modified: Thu, 16 Jun 2011 16:42:57 GMT
```

```
Accept-Ranges: bytes
ETag: "f02a2872442ccc1:3e0f"
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Tue, 16 Aug 2011 12:38:47 GMT

MZ.....................@.....................
.!..L.!This program cannot be run in DOS mode
 (...)
```

Most of the periodic connection with a frequency of one hour were heart beat type of connections such as:

```
GET /ip/ip.asp HTTP/1.1
User-Agent: API-Guide test program
Host: ip.dmy2.com:81
Cache-Control: no-cache

HTTP/1.1 200 OK
Connection: close
Date: Tue, 16 Aug 2011 12:38:40 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 13
Content-Type: text/html
Set-Cookie: ASPSESSIONIDCQSQDCQD=KIEEMPJAAHMEHDJHODABBONA;
path=/
Cache-control: private

147.32.84.165
```

And our analysis conclude that it was not captured because this type of C&C behavior was not seen in the training data. We believe that this C&C behavior can be detected with the proper model.

A.1.9 Description of Scenario 9

This scenario is part of the testing dataset and corresponds to a Neris botnet (MD5 bf08e6b02e00d2bc6dd493e93e69872f) that run on ten bots for 5.18 hours. This is the

same botnet that was used in scenario 1 and 2. The idea of this scenario was to test the synchronicity of the bots by infecting multiple hosts. The main actions performed by the botnet were: to scan the port 3128/TCP on the Internet, to download binary files, to have several C&C channels, to do huge amount of click-fraud, to send SPAM, to connect to hotmail on the web, to connect to google and to connect to yahoo mail. The following paragraphs are the C&C channels on this scenario, since there are ten bots, the C&C channels are ordered by the IP address of the bot:

Label:From-Botnet-V50-*-TCP-CC23-Plain-HTTP-Encrypted-Data These group of labels were assigned to three 4-tuples. First to the 4-tuple *147.32.84.165-195.190.13.78-80-tcp*, , which generated the behavioral state model:

```
53twwwtuwwtwwwwFCfcfwwCcFfFfFwwwcfCcwFwwwwffwwwcwtwtCtwtCFwwwCwtFc
Fcwwwtfttft
```

Second to the 4-tuple *147.32.84.192-195.190.13.78-80-tcp*, which generated the behavioral state model

```
46tFtwwttwwwwwwwwwwFfttwFCttffFwftCwwcCtftwttfFFtFwwwwwFwwFwcwfwcwt
```

Third to the 4-tuple *147.32.84.191-195.190.13.78-80-tcp*, which generated the behavioral state model:

```
46fFcwwtttwwtCwFfFfwFFCtwwcFFwcCcttwttcwwFtwwtffwtwwwwwwwwwwwFvwws
```

These behaviors were really similar in the three bots and consisted in HTTP connections using custom encrypted payloads. They are good representatives of the botnet behaviors. An example of its packets is:

```
GET /blog/footbal.php?ver=303&id=09C164C1CD8F&tick=8647674&
smtp=ok&task=0 HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 195.190.13.78
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Server: nginx/0.7.64
```

```
Date: Wed, 17 Aug 2011 12:54:17 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.2.8

6C3A0C3F303D046F6E6E644933360D373F335D5C6958310836365D5C5C200222
80D3C5B636B6E5B2431023D0A146557646E4D62647A5B654D5561575D530B3E2
5200D300E01650D636A50353E3713605A557600353508 7F352216254D07226B5
(...)
```

Each bot had some differences in the requests parameters and the responses.

Label:From-Botnet-V50-*-TCP-CC5-Plain-HTTP-Encrypted-Data These labels were assigned to the ten bots (ten 4-tuples). To the 4-tuple *147.32.84.165-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr*, to the 4-tuple *147.32.84.191-94.63.150.63-80-tcp*, which generated the model *41rrrrrA*, to the 4-tuple *147.32.84.192-94.63.150.63-80-tcp*, which generated the model *41rrrrrA*, to the 4-tuple *147.32.84.193-94.63.150.63-80-tcp*, which generated the model *41rrrArrr*, to the 4-tuple *147.32.84.204-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr*, to the 4-tuple *147.32.84.205-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr*, to the 4-tuple *147.32.84.206-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr*, to the 4-tuple *147.32.84.207-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr*, to the 4-tuple *147.32.84.208-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr* and to the 4-tuple *147.32.84.209-94.63.150.63-80-tcp*, which generated the model *41rrrrrrr*. This is one of the few behaviors that appeared in all ten bots, connecting to the same C&C server and port. These are fair representatives of the botnet behaviors. An example of its packets is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9ECA82
5FF9F6D9DFE10E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5F44337&v=2&
t=0.3305323 HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180;
Windows NT 5.1.2600)
Host: mewgost.com
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Date: Wed, 17 Aug 2011 14:30:16 GMT
```

```
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch
X-Powered-By: PHP/5.2.6-1+lenny9
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

D2D4AD0619AEF05257BB3D148EF427D3D912DAB2D09FC55F1773084C9248BDE6EEA04AA
E422C2848AEC41A70E0DC739AF90F269D1662DF8F09316D1110BBDB441B9B2989CB6773
(...)
```

The most interesting characteristic of this behavior is that it is not periodic. The requests seems to be done when they are needed. Also the connections do not last long, having an average life time of 2 minutes.

Label:From-Botnet-V50-*-TCP-CC1-HTTP-Not-Encrypted These type of labels were assigned to 28 different 4-tuples from the ten bots because they were similar. The behavioral states models for all the 4-tuples are: *11rr, 11r, 110rr, 11rr, 11r, 110r, 11r, 110r, 11r, 11r, 11r, 11rrr, 11rs, 11r, 11t, 110rr, 11r, 11r, 11rr, 11r, 320rt, 11r, 11rr, 22trs, 22rr, 11t, 11r, , 11rr*. These behaviors correspond to C&C channels that send only two or three flows and use the HTTP protocol without any encryption. They are an example of a C&C channel with few information that is not periodic. This is one of the few behaviors that appeared in all ten bots, connecting to the same C&C server and port. They are bad representatives of the botnet behaviors. An example of its packets are:

```
POST /?fba5a9a1bee8a5aaadacabaebeeda5aea1aeafbeeea5abacaba8befaa5a8befca5a
9beeba5a9aaa0acaca8aebefda5dba2c4cfd1d6dcd7cfcbc4ccddd5c8c4f1feabfcededfcf
2ebb6fde0fdbeffa5f1f6f1ec98 HTTP/1.1
Accept: */*
Accept-Language: en-us
CB2: 1
Accept-Encoding: gzip, deflate
User-Agent: Mozilla
Host: 202.112.126.218


TTP/1.1 200 OK
Date: Wed, 17 Aug 2011 12:32:28 GMT
Server: Apache/2.2.8 (Fedora) DAV/2 PHP/5.2.6 mod_ssl/2.2.8
```

```
OpenSSL/0.9.8g
X-Powered-By: PHP/5.2.6
Content-Length: 26
Connection: close
Content-Type: text/html; charset=UTF-8

CB2=212.117.171.138:65500
```

Label:From-Botnet-V50-*-TCP-CC16-HTTP-Not-Encrypted These labels were assigned to the ten bots (ten 4-tuples), which generated the following behavioral state models:

```
33CCatttrrttttAttttatttattrttttcatttttrwtttcattttcswtcttcttttBwtcttAtttAttttattt
tattttattrttttcsttCtCAwtttCrttttCArwttttrwttttattttstttFAttttstttttttsttt,
33tstttrttrttttCawtCttrrttttbttttCAttttCattttCsttCcattttAttttatttrttrwtttcsttcttB
ttttCswttcsrwttttstttCrwtccrttctrttttttttAttttrttttc,
33crttrttCtcAttttstttCawttttAttCtCAwtttCatttttrtttcCattCCttttrttttrttrttttttrttttt
BtttCawttttrwttttrwttttttttCswtCcatttrttttAwttrwtttAwttttt,
33rwtttcrttttttrttttsrttttCsttcCcAttttcBttttAttttbttttAtttrttttCAtttcrwtttrwtttcrr
wttttrwtttcBrwtttrwtttCAttccAttttcAwtttcBwtCccB,
63ttcrwtttCBwttttAsttttcBttcttsrttttbrttttrwtttAtwtttttrwtttctwttCtattttatttrttBrt
ttCtattttAtttttcCAttttrttCtrttttttrwtttC,
63ttcsttttcsttttttswCtctrwttttsttcttswttttrwctttrwtCttrttttrttttttrttrttttttrwtCttrCtt
ttrwtttsttttttrFttttttttctsttCtrwttttrtCttrttttsttCtswttttrFt,
33ttCAwtttCrtttttAttcCtawtttCatttCtAttttrttttBttttbttttrttttrttttcAwtttcrttttttsttct
trwttttsttttttsttttttsttttttAsrACtttrttttttrttttr,
63tCtswttttrwttCtsttttttrwtcttswttttrrttCtrrttttattrwttwtrwttttstcCttrwttttwscttrtt
tttsrttttrttttrrttttsttttrwttttrr,
11rrfttttswttttrttttrttttrttttrttttrttttrttttttsrwtCttrwttttsttttttsttttttsttttttrtttrtt
ttsrrCtttrwtttt,
13ttctAwtCtcBwtCtcsrttttrttttattttattttrtttrttrttttttsttCctawtttcrwtttbAwtttCBttttt
rttttCawtcCCttttAttttAttttAttttrwtttcrw
```

These C&C channels use the HTTP protocol with a custom encryption. These are good representatives of a C&C channel. This is one of the few behaviors that appeared in all ten bots, connecting to the same C&C server and port. They are good representatives of the botnet behaviors. An example of its packets is:

```
GET /g1.php?u=hhhh&v=6&m=0BADFF24-D7B3-4A25-A789-317E03ACFA77 HTTP/1.1
```

```
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; waoc)
Host: www.lddwj.com
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Wed, 17 Aug 2011 12:29:10 GMT
Server: Apache/2.2.19 (Unix) mod_ssl/2.2.19 OpenSSL/0.9.8e-fips-rhel5
mod_auth_passthrough/2 mod_bwlimited/1.4 FrontPage/5.0.2.2635
X-Powered-By: PHP/5.2.17
Transfer-Encoding: chunked
Content-Type: text/html

3
...
JBFXDEHYEXJM
-----
EUHYBOIQGJEOILGWELFR
-----
JHGKBNBIIVENGPGCGFCTFJFIFAJDEUIIHI
46
JHGKBNBIIVENGSBUBWCXFUCQBEHJEFAUEUDBHEGFDSDABNHHEXBMAQHKIBASBAAEAVIJ
3e
JHHBEXJDBRGTISJCBWJKDSJKBYGJGTDVCCBVIQIYGBEHFXEBGZBWGGDOBAIO
29
JHGKBNBIIVENGMJJJQAVFKAKETAGGDHQ
-----
1f
DUHNHLCSCEGRDPJQAYDJBB
-----
3c
JHHBEXJDBRHIIHGJCDCQHUHHHTBWAEJPAYCXJSDKCYGNDUDLDJGGIYHOGF
(...)
```

Label:From-Botnet-V50-*-TCP-CC6-Plain-HTTP-Encrypted-Data These labels were assigned to the ten bots (ten 4-tuples, such as 147.32.84.191-60.190.223.75-888-tcp), which

generated the following behavioral state models: *11rrrArrrrAArr, 11ArrrrrrrArrr, 11rArrrssraArr, 11rABrrrrrrrrr, 11rrrrrrrrArrr, 11rrrrArrrrrrr, 11rrrrrrrrAArr, 11rrrrrrrrAArr, 11rArrsrrrBArs, 11rrrasrrrAAAA*. These C&C channels use the HTTP protocol to send probably encrypted data. This is a good example of a C&C. An example of its packets is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9ECA825F
F9F6D9DFE10E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5FF432C&v=2&t=0.8972437
HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180; Windows NT 5.1.2600)
Host: w.nucleardiscover.com:888
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Connection: keep-alive
Date: Wed, 17 Aug 2011 00:07:45 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-Powered-By: Zend Core/2.5.0 PHP/5.2.4
Content-Length: 2048
Content-Type: text/html

12149A31890C43E3A44EDBF6A1E931C5EE77A6C587C8D14D94A6B5FDA57FAF801259ACBA45
2ABADBB98CBFD4D5DB170F53A96AD4CEBB072F8BB9D8A5ABD7ACC7C986F454D27EEB68A0E2
B03BD5223DE286A9901104C6B083EC46F5969C7EE04B6BE63B3399FB3DEE29DD2AA2777D4B
(...)
```

Label:From-Botnet-V50-1-TCP-CC12-HTTP-Not-Encrypted These labels were assigned to the ten bots (ten 4-tuples such as 147.32.84.165-91.220.0.52-80-tcp), which generated the following behavioral state models:

```
13yrAtrxx0z, 11rCrrcacstrraaAracrrBsBbaAaarCrraaAAAAaACttrssaCsrAaaaCtcara
ctrsrrBBAABcrrrraCtIrstz,11rAaaaaaActsCrrcstsrBCrrCsrtrrBaBCrrCrrrtrtrrssA
Acstrztstz, 11rctararrtrtrtssrttrrtrtstrrraAbCrrrtsrsrttrrCrtrraarrriziztz
tzz, 18ttgtttCarcrrCrrAaAAAaaCrtrrACrrrsCrzzIIsszz, 31rrrrrrrrzrrrtrtrrtrr
strrrtrstttrrrtrrtrrszzrrrz, 17rstzr, 33trrrtrstttrrrssrrtsrrtrtrrrrrrtrrt
rtrsrsrsAAaBBACzzrztyz, 17xggAaaAaAacrrAaCtasCrsAAaAcstAsAraCrraartrtrrtss
```

```
zrrrt, 33rrrrrrrtsrrrrrrrrtttrrrtrrrrrrtrrrrrrtszzzzzzzssttAsrAbrBCtssAArb
AAaabBAtaaAaBttasrraabAAaAAAraBrraaacrraBrttArCtrsAAbsaABrAcrsBrrABbbbaAab
ABrt
```

Interesting enough the models look alike but they are not so similar. However the content of the HTTP C&C is very similar and the purpose seems to be the same. They are a good example of a C&C connection. An example of its packets is:

```
POST /ajax.php HTTP/1.1
Host: 897234kjdsf4523234.com
Content-Length: 223
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Connection: close

9f94ee2e06c3cb29=c0SZ%2B2rm3YICP%2BKokLJvSVltAPiUiVPlM1apXN6cv5VRUJP7wHje
WUqZ1rEqtFEF%0D%0Aw%2BTSsqgXyxFW7uhY%2Fv%2Fic%2FebqKYI%2BYwlXBcXn1tXRerMe
gl8etc7qmRu7uH7R7V0%0D%0AE86IH1K40W8Zxte1xhvC70o1NnjffqSs4oFKR1Lz1S8%3D
%0D%0A

HTTP/1.1 200 OK
Server: nginx/1.1.0
Date: Wed, 17 Aug 2011 12:26:31 GMT
Content-Type: text/html
Connection: close
Content-Length: 25

5gCUo0nG65lhMmqKq6YPwQ==
```

Label:From-Botnet-V50-4-TCP-CC20-Plain-HTTP-Encrypted-Data These labels were assigned only to seven bots (seven 4-tuples such as 147.32.84.193-195.190.13.70-80-tcp), which generated the following behavioral state models:

```
46wwftttwtwwwtcFCwtFCFtwwFFfCwwttwwwtFtccwtwFFctFtwtcwftttfwCctCtwts
46FwwwwwwFtctCtwFttwwcfCFwttCwFcwFtwwwwttwwCttwwtwtttwCwwtccwwwtws
43CFfwtwtttFwfwwwtwCFFwfttCfCCFCwtttwFFfwtFwtFwtwwtCftffFCwwtttwCt
46fwwcCCtttwtwwtttfttFFwwfwwCwCcwwwCFtFCwwwCwFCftwtfFwwwfcwtwtCv
43wtwftwFwwwtCttftwwctwtwwtCCCfFwftFFwtfttttffwwwwCwwftwwCs
43wtttftfFctwtcFttCtwtwtwwttwtttFwcwtfttCCCcttwcwwtfctwFwtwwts
46twtCtCftFwwttwwfFfffFctttttCwtCfFtwwwCtwcCttwfFCtwwwwwtwttw
```

These behaviors are really similar and they use the HTTP protocol to send probably encrypted data. These are good example of a C&C channel. An example of its packets is:

```
GET /topic/supra.php?ver=303&id=09C164C1CD8F&tick=9282848&smtp=ok&task=10&
errors[0]=0&errors[702]=3&errors[703]=97 HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 195.190.13.70
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Server: nginx/0.6.36
Date: Wed, 17 Aug 2011 16:07:56 GMT
Content-Type: text/html
Connection: keep-alive
X-Powered-By: PHP/5.2.5
Content-Length: 5046

6C3A0C3F303D046F6E6E644933360D373F335D5C6958310836365D5C5C200222080D3C5
B61696E5B2431023D0A146557646E4D62647A5B654D5561555D530B3E25200D300E0165
57646E4D62647A5B654D5561555D5310252F38066C5369525A7F300D37396A6E5B5F013
(...)
```

A.1.10 Description of Scenario 10

This scenario is part of the training dataset and corresponds to a Rbot botnet (MD5 2467b3c8b259cecd6ce2d5c31009df10) that run on ten bots for 4.75 hours. This version of the Rbot botnet is a custom compilation, so we can control the IRC server. The idea of this scenario is to perform controlled DoS attacks and capture their behavior. The actions of the bot were: to use an IRC C&C channel, to get information about the bots, to perform a UDP DoS attack, to perform a ICMP DoS attack and to perform a TCP DoS attack. The following paragraphs are the C&C channels of this scenario:

Label:From-Botnet-V51-10-TCP-CC77-IRC-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.209-213.232.93.3-6667-tcp*, which generated the behavioral state model *990t*. This C&C sent few flows because the IRC protocol uses the same connection with a heart beat. This is a good example of an IRC C&C. An example of its packets is:

```
MODE Pepe971541 -x
JOIN #zarasa48
USERHOST Pepe971541

#zarasa48 :.sysinfo
PRIVMSG #zarasa48 :System [CPU]: 3200MHz. [RAM]: 1,048,048KB
total, 1,048,048KB free. [Disk]: 52,420,060KB total, 48,792,296KB free.
[OS]: Windows XP (Service Pack 2) (5.1, Build 2600). [Sysdir]:
C:\WINDOWS\system32. [Hostname]: saruman (147.32.84.165).
[Current User]: Ctu. [Date]: 18:Aug:2011. [Time]: 12:08:17.
[Uptime]: 0d 0h 21m

PRIVMSG #zarasa48 :System [CPU]: 3166MHz. [RAM]: 523,760KB total,
523,760KB free. [Disk]: 52,420,060KB total, 48,810,320KB free.
[OS]: Windows XP (Service Pack 2) (5.1, Build 2600). [Sysdir]:
C:\WINDOWS\system32. [Hostname]: saruman2 (147.32.84.192).
[Current User]: Ctu. [Date]: 18:Aug:2011. [Time]: 12:08:18. [Uptime]: 0d 0h 13m

PRIVMSG #zarasa48 :.udpflood 147.32.96.69 100000 1500 10 161
PRIVMSG #zarasa48 :.synflood 147.32.96.69 1 1000
PRIVMSG #zarasa48 :.icmpflood 147.32.96.69 1000
```

A.1.11 Description of Scenario 11

This scenario is part of the training dataset and corresponds to a Rbot botnet (MD5 2467b3c8b259cecd6ce2d5c31009df10) that run on three bots for 0.26 hours. This version of the Rbot botnet is a custom compilation, so we can control the IRC server. The idea of this scenario is find out how many bots are necessary to take down another host in the Internet using aDDoS attack. Our experiments confirmed that only two bots with a good bandwidth can be used to make an ICMP DoS attack and take down another host with a good bandwidth . The actions of the bots were: to use an IRC C&C channel, to get information about the bots, to perform a UDP DoS attack, to perform a ICMP DoS attack and to perform a TCP DoS attack.

The goal of this scenario was to make a DoS attack, but only using the fewer amount of packets possible in the C&C channel. Therefore, each infected host connected to the IRC channel, received a single order and attacked. The results were a total amount of 63 C&C packets on the channel for each bot. These 63 packets were converted to only one flow for each bot, resulting that there were not enough flows to generate a single

4-tuple. This is an example of a very short C&C channel doing a real attack and may be used as the baseline minimum malicious action.

Our method did not generate any single behavior from this experiment.

A.1.12 Description of Scenario 12

This scenario is part of the training dataset and corresponds to a NSIS.ya botnet (MD5 eaf85db9898d3c9101fd5fcfa4ac80e4) that run for 1.21 hours on three bots. The main actions of the botnet were: to access several search web pages suchas baidu and google, to access several general-type web pages, to download binary files and to connect to a P2P C&C channel. The following are the C&C channels on this scenario:

Label:From-Botnet-V*-UDP-CC108-Established-P2P-Not-Encrypted-1 These labels were assigned to six 4-tuples, such as *69.104.66.134-147.32.84.192-31037-udp*, which generated the behavioral state models *11rr, 13rww, 11sr, 11rr, 11rr, 12rr*. The behavior is short because the P2P network was not contacted for a long time, only for 1.21 hours. This is a good example of a C&C channel, but it was too short. An example of the strings on its packets is:

```
d1:ad2:id20:
:info_hash20:
e1:q9:get_peers1:t8:
1:y1:qe
d1:rd2:id20:
e5:nodes208:
d1:ad2:id20:
6:target20:
d1:rd2:id20:
e5:nodes208
d1:ad2:id20:
6:target20:
Re1:q9:find_node1:t4:
51:v4:UTb*1:y1:qe
d1:ad2:id20:
9:info_hash20:
4:porti9306e
5:token20:
```

```
e1:q13:announce_peer1:t8:
(...)
```

A.1.13 Description of Scenario 13

This scenario is part of the training dataset and corresponds to a Virut botnet (MD5 85f9a5247afbe51e64794193f1dd72eb) that run for 16.36 hours in one bot. The main actions of the botnet were: . The following paragraphs are the C&C channels of the botnet:

Label:From-Botnet-V54-1-TCP-CC12-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-91.220.0.52-80-tcp*, which generated the behavioral state model *18rbrrrrzxxyryrraAAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa*. It uses the HTTP protocol with a probable encrypted payload. It is good example of a periodic C&C channel. An example of its packets is:

```
POST /ajax.php HTTP/1.1
Host: 897234kjdsf4523234.com
Content-Length: 217
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Connection: close

HTTP/1.1 200 OK
Server: nginx/1.1.0
Date: Mon, 15 Aug 2011 15:21:30 GMT
Content-Type: text/html
Connection: close
Content-Length: 25

QcyviUxaWsUwEv1ylI5liw==
```

This is the same behavioral pattern of other CC12 labels in previous scenarios.

Label:From-Botnet-V54-1-TCP-CC5-Plain-HTTP-Encrypted-Data This label was assigned to the 4-tuple *147.32.84.165-94.63.150.63-80-tcp*, which generated the behavioral state model *41rrrrr0uu0u0Drr0u0d*. It is a fair representative of the botnet behaviors. An example of its packets is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9ECA82
5FF9F6D9DFE10E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5F44337&
v=2&t=0.3921167
HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180;
Windows NT 5.1.2600)
Host: mewgost.com
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Date: Mon, 15 Aug 2011 17:26:01 GMT
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch^
X-Powered-By: PHP/5.2.6-1+lenny
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

1B1DE14A75F640E15BBC351B5B2C7DBD9B02E7DC2C0A1388187A6726A0050026205EC72D
107A0658BA849B045C4E0AEDC500BD55D7A4391D4524730705AF600F542A7BDBA10DB033
(...)
```

Label:From-Botnet-V54-1-TCP-CC6-Plain-HTTP-Encrypted-Data This label was assigned to the 4-tuple *147.32.84.165-60.190.223.75-888-tcp*, which generated the behavioral state model *120tttCtCCtttCttt0rrrrrrAAAA0rr0rr*. It uses the HTTP protocol with probably encrypted payloads. It is a good example of a C&C channel. An example of its packets is:

```
GET /list.php?c=B4AC885F94224AE64DAAC6F60346C27CD049B58C0B2469F2DC9ECA825FF9
F6D9DFE10E13F3845D3386FFC45E0D4897B5778D4CBB9FE6A5F44337&v=2&t=0.3921167
HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0.2900.2180; Windows NT 5.1.2600)
Host: mewgost.com
Connection: Keep-Alive
Pragma: no-cache

HTTP/1.1 200 OK
Date: Mon, 15 Aug 2011 17:26:01 GMT
```

```
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch
X-Powered-By: PHP/5.2.6-1+lenny9
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

1B1DE14A75F640E15BBC351B5B2C7DBD9B02E7DC2C0A1388187A6726A0050026205EC72D107
A0658BA849B045C4E0AEDC500BD55D7A4391D4524730705AF600F542A7BDBA10DB033B0F2FC
772DDA4B94E3CC59D85E9CFCCFC16BFE9DFC1EE150C6480C31E8B08156A85844D68B756E3F0
(...)
```

Label:From-Botnet-V54-1-TCP-CC16-HTTP-Not-Encrypted This label was assigned to the 4-tuple *147.32.84.165-173.192.170.88-80-tcp*, which generated the behavioral state model:

```
31trtrtttstrttwtsrtrtrtrtrtrtrtrtrtttttCCtrtrtrtrtrtrtttrtrtrtttttttttrtr
trtrtrtrtrtrtrtrtrtrtttttttrtrtrtrtrtrtrtrtrtrtrtrtttttttrtrtrtrtrtrtrtrtrt
rtttttttrtrtrtrtrtrtrtrtrtrtrtrtttttttttrtrtrtrtrtrtrtrtttrtrttttrtrtrtrtrt
rtrtrtrtrtttrtCtctttttbtttrtrtrtrtrtrtrtrtrtrtrtrtrtrtrtstrtrtrtrtrtrtr
trtrtrtrtrtrtrtrtrrtrtsttttttttttttttttttttttccttttttttttttcCtttctttwtttt
tttCtttttwttttttttttttttttrtttttttrtrttrrrstsrttsttrsCttttsttctttttttttcC
ttsttt
```

This C&C channel uses the HTTP protocol with probably encrypted payloads. It is a good example of a C&C channel. An example of its packets is:

```
ET /g1.php?u=hhhh&v=6&m=36456E29-4303-4E66-8DAC-DBD246E55B62 HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; waoc)
Host: www.lddwj.com
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Mon, 15 Aug 2011 17:48:54 GMT
Server: Apache/2.2.19 (Unix) mod_ssl/2.2.19 OpenSSL/0.9.8e-fips-rhel5 mod_auth_passth
X-Powered-By: PHP/5.2.17
Transfer-Encoding: chunked
```

```
Content-Type: text/html

3
...
JBFXDEHYEXJM
-----
EUHYBOIQGJEOILGWELFR
-----

24
JHGKBNBIIVENGPGCGFCTFJFIFAJDEUIIHI

46
JHGKBNBIIVENGSBUBWCXFUCQBEHJEFAUEUDBHEGFDSDABNHHEXBMAQHKIBASBAAEAVIJ

3e
JHHBEXJDBRGTISJCBWJKDSJKBYGJGTDVCCBVIQIYGBEHFXEBGZBWGGDOBAIO
```

Appendix B

Implementation of the algorithms

This appendix briefly describes the features of the free software tools developed as part of the implementation of our detection algorithms. Also there are some complementary tools that may be useful for the research community. All the tools were published under the GPLv2 license to encourage the free distribution and improvement of the ideas. We believe that publishing our detection method as a free software tool could greatly improve the botnet detection area.

B.1 CCDetector

The CCDetector.py tool corresponds to the main implementation of the CCDetector method. It was done completely using the Python language. The published implementation actually has the same thresholds of the state-based model that were used during this thesis. Any researcher can use it without any modifications. The documentation in the public git repository shows how to install and use it. This program includes all the processing stages of our algorithm and is capable of generating the state-based model, of training new models of behavior, of monitoring traffic in real-time and of generating the final NetFlow file with the predicted labels. The last version can be downloaded from [1].

The CCDetector.py tool comes ready to process the network traffic. Some of the parameters are:

- -f (–file) : Input NetFlow file to analyze. If - is used, NetFlows are read from stdin. The file should be space separated.

[1] https://bitbucket.org/eldraco2000/ccdetector

- -T (–analyze-tuple) : Analyze only this tuple and print detailed information for each netflow.
- -r (–training) : Training mode. Read NetFlows and outputs one Markov Chain for each label in the folder 'MCModels'.
- -e (–testing) : Testing mode. Read NetFlows from -f file, Markov Chains from the folder 'MCModels', predicts for each tuple the chain (label) with more probability of generating it and it outputs a labeled NetFlow file.
- -p (–prob-threshold) : Threshold to use when comparing each tuple to every model.
- -L (–label) : Print all the informatin about all the tuples with this label.
- -P (–print-mode) : Print mode: normal, csv, oneline, epoch and live. You can combine them with '-'. Live means ncurses live.

An example usage of the CCDetector.py tool to generate the botnet models and to generate a labeled NetFlow file is:

- Capture some traffic and store it in pcap format, such as train.pcap.
- Convert the traffic to Argus format with the command: `argus -F argus.conf -r train.pcap -w train.biargus`. Make sure that the Argus flows are bidirectional in the configuration.
- Manually label the NetFlows using the ralabel tool. The rules for your labels should go into some file that the ralabel configuration file knows where it is. The ralabel command is: `ralabel -f ralabel.conf -r train.biargus -w train.biargus.labeled`
- Obtain the labeled NetFlow file with the ra tool. The command is: `ra -F ra.conf -n -Z b -r train.biargus.labeled > train.binetflow`. The ra.conf file should have the following format line: `RA_FIELD_SPECIFIER= stime dur proto:10 saddr:27 sport dir daddr:27 dport state stos dtos pkts bytes sbytes label:40`.
- Generate the training models with the CCDetector.py tool. The command is : `CCDetector.py -f train.binetflow -r`. The result is a folder called MCModels with all the state-based behavioral models for each label.
- Capture some unknown traffic in a pcap file, such as test.pcap.
- Convert the pcap file to biargus as above.

- Label the biargus file as above if you want to have error metrics.
- Generate the binetflow file with the ra tool as above.
- Detect similar behavior in new traffic with the CCDetector.py tool. The command is: `CCDetector.py -f test.binetflow -e`. The tool will automatically read the MCModels folder looking for the models. The results will be a test.binetflow.labeled file that has the predictions as a new column.
- Error metrics. To compute the error metrics see the Botnet Detectors Comparer tool.

B.2 Botnet Detectors Comparer

This tool is our implementation of the new error metrics and comparison methodology. When the final NetFlow file is generated by the CCDetector tool, there is no easy way of computing how well the prediction was. The Botnet Detectors Comparer tool is designed to read these NetFlow files with predictions and to output the error metrics. This tools is the basis of our comparison methodology of botnet detection methods. It is already prepared to use the same thresholds that our work so it is easier to compare a third-party method with our dataset. This tool implements all the new definitions of error metrics described in 7.4. The last code can be found online [2].

B.3 Malware Database Administrator: Madab

For the purpose of managing all the malware downloaded in the Malware Capture Facility Project the CTU student Vojtěch Uhlíř developed a malware binaries management database. This tool is designed to keep a record of all the malware being used, assigning tags and allowing for a fast access. It was crucial in the organization of the MCFP. The software is published as free software for the community [3].

[2] https://sourceforge.net/projects/botnetdetectorscomparer/
[3] https://github.com/WojtylaCZ/MalwareDB

Bibliography

[1] Chester Wisniewski. Exposing the Money Behind the Malware. Technical report, Shopos, 2012.

[2] Richard Ford and Sarah Gordon. Cent, five cent, ten cent, dollar: hitting botnets where it really hurts. In *NSPW '06: Proceedings of the 2006 workshop on New security paradigms*, pages 3–10, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-923-4. doi: http://doi.acm.org/10.1145/1278940.1278942.

[3] Andrzej Kozlowski. Comparative Analysis of Cyberattacks on Estonia, Georgia and Kyrgyzstan. *European Scientific Journal*, 3:237–245, 2014. URL http://eujournal.org/index.php/esj/article/view/2941.

[4] Lawrence E Menten, Aiyou Chen, and Dimitrios Stiliadis. NoBot : Embedded Malware Detection for Endpoint Devices. *Bell Labs Technical Journal, John Wiley & Sons, Ltd*, 16(1):155–170, 2011. doi: 10.1002/bltj.

[5] David Zhao, Issa Traore, Ali Ghorbani, and Bassam Sayed. Peer to Peer Botnet Detection Based on Flow Intervals. In *Information Security and Privacy Research IFIP Advances in Information and Communication Technology*, volume 3, pages 87–102. Springer, 2012. ISBN 978-3-642-30436-1. doi: 10.1007/978-3-642-30436-1_8. URL http://link.springer.com/chapter/10.1007/978-3-642-30436-1_8.

[6] Yong Qiao, Yue-xiang Yang, Jie He, Chuan Tang, and Ying-zhi Zeng. Detecting P2P bots by mining the regional periodicity. *Journal of Zhejiang University SCIENCE C*, 14(9):682–700, September 2013. ISSN 1869-1951. doi: 10.1631/jzus.C1300053. URL http://link.springer.com/10.1631/jzus.C1300053.

[7] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, Wei Lu, J. Felix, and P. Hakimian. Detecting P2P Botnets through Network Behavior Analysis and Machine Learning. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, pages 174 – 180, Montreal, Canada, 2011. IEEE. ISBN 9781457705847. doi:

10.1109/PST.2011.5971980. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5971980&isnumber=5971950.

[8] Sebastian Garcia, Alejandro Zunino, and Marcelo Campo. Survey on Network-based Botnet Detection Methods. *Security and Communication Networks, John Wiley & Sons, Ltd*, 7(5):878–903, 2013. doi: 10.1002/sec.800. URL http://onlinelibrary.wiley.com/doi/10.1002/sec.800/abstract.

[9] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. *2012 IEEE Symposium on Security and Privacy*, pages 65–79, May 2012. doi: 10.1109/SP.2012.14. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6234405.

[10] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali a. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, May 2012. ISSN 01674048. doi: 10.1016/j.cose.2011.12.012. URL http://www.sciencedirect.com/science/article/pii/S0167404811001672.

[11] DP Anderson. Boinc: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburg, 2004. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1382809.

[12] Timothy B. Lee. How a grad student trying to build the first botnet brought the Internet to its knees, November 2013. URL http://wapo.st/1nW43gd.

[13] Dennis Fowler. Microsoft's Trials and Tribulatins. *Net News*, (September):7–12, 1999.

[14] John Canavan. The evolution of malicious IRC bots. Technical report, Symantec, 2005.

[15] L McLaughlin. Bot software spreads, causes new worries. *Distributed Systems Online, IEEE*, 5(6):1–5, 2004. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1308173.

[16] D Dittrich and S Dietrich. P2P as botnet command and control: a deeper insight. In *3rd International Conference on Malicious and Unwanted Software*, pages 41–48, 2008. doi: 10.1109/MALWARE.2008.4690856. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4690856.

[17] Junjie Zhang, Roberto Perdisci, and Wenke Lee. Detecting stealthy P2P botnets using statistical traffic fingerprints. In *Dependable Systems & Networks*, pages 121—-132. IEEE Computer Society, 2011. doi: 10.1109/DSN.2011.5958212. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958212.

[18] T Holz. A short visit to the bot zoo. *Security & Privacy, IEEE*, pages 76–79, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1439508.

[19] I Arce and Elias Levy. An analysis of the slapper worm. *Security and Privacy, IEEE*, (September 2002), 2003. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1177002.

[20] Stefano Schiavoni, F Maggi, L Cavallaro, and Stefano Zanero. Tracking and Characterizing Botnets Using Automatically Generated Domains. *Cornell University Library arXiv*, 2013. URL http://arxiv.org/abs/1311.5612.

[21] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. FluXOR : Detecting and Monitoring Fast-Flux Service Networks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Lecture Notes in Computer Science, pages 186–206. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-70541-3.

[22] Yacin Nadji, M Antonakakis, and R Perdisci. Beheading hydras: performing effective botnet takedowns. In ACM New York, editor, *CCS '13 Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 121–132. ACM New York, 2013. ISBN 9781450324779. doi: 10.1145/2508859.2516749. URL http://dl.acm.org/citation.cfm?id=2516749.

[23] Jason Franklin and Adrian Perrig. An inquiry into the nature and causes of the wealth of internet miscreants. In ACM, editor, *CCS '07 Proceedings of the 14th ACM conference on Computer and communications security*, pages 375–388. ACM, 2007. ISBN 9781595937032. URL http://sparrow.ece.cmu.edu/group/pub/franklin_paxson_perrig_savage_miscreants.pdf.

[24] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *SEC'11 Proceedings of the 20th USENIX conference on Security*, pages 13–13, 2011. URL https://www.usenix.org/legacy/events/sec11/tech/full_papers/Caballero.pdf.

[25] Microsoft. Microsoft Security Intelligence Report. Technical report, 2013. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Microsoft+Security+Intelligence+Report#1.

[26] H R Zeidanloo and A B A Manaf. Botnet Detection by Monitoring Similar Communication Patterns. *International Journal of Computer Science and Information Security*, 7(3):36–45, 2010. ISSN 19475500. URL http://sites.google.com/site/ijcsis/.

[27] Anirudh Ramachandran and Nick Feamster. Understanding the network-level behavior of spammers. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 291–302, New York, NY, USA, 2006. ACM. ISBN 1-59593-308-5. doi: http://doi.acm.org/10.1145/1159913.1159947.

[28] A. M. Konovalov, I. V. Kotenko, and A. V. Shorov. Simulation-based study of botnets and defense mechanisms against them. *Journal of Computer and Systems Sciences International*, 52(1):43–65, February 2013. ISSN 1064-2307. doi: 10.1134/S1064230712060044. URL http://link.springer.com/10.1134/S1064230712060044.

[29] Sebastian Garcia, Alejandro Zunino, and Marcelo Campo. Botnet Behavior Detection using Network Synchronism. In Peyman Kabiri, editor, *Privacy, Intrusion Detection and Response: Technologies for Protecting Networks*, pages 122–144. IGI Global, Pensilvania, 2012. ISBN ISBN13: 9781609608361, ISBN10: 1609608364, EISBN13: 9781609608378. doi: 10.4018/978-1-60960-836-1.ch005. URL http://sebastian-garcia.isistan.unicen.edu.ar/publications/garciachap_kabiribook.pdf?attredirects=0.

[30] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An Empirical Comparison of Botnet Detection Methods. *Computers & Security*, 45(0):100 – 123, 2014. ISSN 0167-4048. doi: http://dx.doi.org/10.1016/j.cose.2014.05.011. URL http://www.sciencedirect.com/science/article/pii/S0167404814000923.

[31] Martin Rehak, Michal Pechoucek, and Pavel Celeda. CAMNEP: agent-based network intrusion detection system. In *AAMAS '08 Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 133–136, Estoril, Portugal, 2008. International Foundation for Autonomous Agents and Multiagent Systems. URL http://dl.acm.org/citation.cfm?id=1402820.

[32] 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*, pages 1–16. USENIX Association, 2007. URL http://portal.acm.org/citation.cfm?id=1362903.1362915.

[33] Andrew Conway. Twenty Years of Spam, 2014. URL http://blog.cloudmark.com/2014/04/11/twenty-years-of-spam/.

[34] Zhaosheng Zhu, Guohan Lu, Yan Chen, Zhi Judy Fu, Phil Roberts, and Keesook Han. Botnet Research Survey. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 967–972. Ieee, July 2008. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4591703.

[35] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A survey of botnet technology and defenses. In *Cybersecurity Applications & Technology Conference For Homeland Security*, pages 299–304, California, 2009. Ieee. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4804459.

[36] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 268–273. Ieee, 2009. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5210988.

[37] Elizabeth Stinson and J.C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *Proceedings of the 2nd conference on USENIX Workshop on offensive technologies*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association. URL http://portal.acm.org/citation.cfm?id=1496707.

[38] V.C. Estrada and A. Nakao. A Survey on the Use of Traffic Traces to Battle Internet Threats. In *2010 Third International Conference on Knowledge Discovery and Data Mining*, pages 601–604. Ieee, January 2010. ISBN 978-1-4244-5397-9. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5432468.

[39] Jing Liu, Yang Xiao, Kaveh Ghaboosi, Hongmei Deng, and Jingyuan Zhang. Botnet: Classification, Attacks, Detection, Tracing, and Preventive Measures. *EURASIP Journal on Wireless Communications and Networking*, 2009:1–12, 2009. ISSN 1687-1472. URL http://mts.hindawi.com/utils/getacceptedmsfile.aspx?msid=692654&vnum=2&ftype=manuscript.

[40] Chao Li, Wei Jiang, and Xin Zou. Botnet: Survey and Case Study. In *2009 Fourth International Conference on Innovative Computing Information and Control ICICIC*, volume 0, pages 1184–1187. Ieee, 2009. ISBN 9781424455430. doi: 10.1109/ICICIC.2009.127. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5412718.

[41] TrendMicro. Taxonomy of botnet threats. Technical report, Trend Micro, 2006. URL http://emea.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf.

[42] S. Y. Lim and A. Jones. Network Anomaly Detection System: The State of Art of Network Behaviour Analysis. *Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on*, pages 459–465, 2008.

[43] Amit Kumar Tyagi and G Aghila. A Wide Scale Survey on Botnet. *International Journal of Computer Applications*, 34(9):9–22, 2011.

[44] Mohammad Jorjor Shooshtari Zadeh, Hossein Rouhani Zeidanloo, M. Safari, Mazdak Zamani, and Payam Vahdani Amoli. A Taxonomy of Botnet Detection Techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, pages 158–162. IEEE, 2010. ISBN 9781424455379. doi: 10.1109/ICCSIT.2010.5563555. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5563555.

[45] G Gu, J Zhang, and W Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network traffic. In *Proc. 15th Network and Distributed System Security Symposium (NDSS), San Diego, CA, February*, 2008.

[46] Wei Lu, Goaletsa Rammidi, and Ali A Ghorbani. Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, 34(3):502–514, 2010. ISSN 0140-3664. doi: 10.1016/j.comcom.2010.04.007. URL http://www.sciencedirect.com/science/article/B6TYP-4YWC159-1/2/ea30180085250d194d113823a5f7a4ce.

[47] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association.

[48] Z. Li, B Wang, D Li, H Chen, F Liu, and Z.B. Hu. The Aggregation and Stability Analysis of Network Traffic for Structured-P2P-based Botnet Detection. *Journal of Networks*, 5(5):517–526, 2010. ISSN 1796-2056. URL http://academypublisher.com/ojs/index.php/jnw/article/viewArticle/0505517526.

[49] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, C. Kruegel, and E. Kirda. *Automatically generating models for botnet detection*, pages 232–249. Springer, Berlin, 2010. ISBN 978-3-642-04443-4. URL http://www.springerlink.com/index/64U6455075084115.pdf.

[50] M Patrick Collins, Timothy J Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. Using uncleanliness to predict future botnet addresses. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 93–104, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-908-1.

[51] W. Timothy Strayer, R. Walsh, C Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 195–202, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4116547&tag=1.

[52] Ting-Fang Yen and Michael Reiter. Traffic Aggregation for Malware Detection. In Springer Berlin / Heidelberg, editor, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 5137/2008 of *Lecture Notes in Computer Science*, pages 207–227. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-70541-3.

[53] Xiaocong Yu, Xiaomei Dong, Ge Yu, Yuhai Qin, Dejun Yue, and Yan Zhao. Online Botnet Detection Based on Incremental Discrete Fourier Transform. *Journal of Networks*, 5(5):568, May 2010. ISSN 1796-2056. URL http://ojs.academypublisher.com/index.php/jnw/article/view/853.

[54] DH Kim, Taek Lee, Jaewoo Kang, and Hyunchoel Jeong. Adaptive pattern mining model for early detection of botnet-propagation scale. *Security and Communication Networks, John Wiley & Sons, Ltd*, 2011. doi: 10.1002/sec. URL http://onlinelibrary.wiley.com/doi/10.1002/sec.366/full.

[55] Marcus A. Maloof. Some Basic Concepts of Machine Learning and Data Mining. In Marcus A. Maloof, editor, *Machine Learning and Data Mining for Computer Security Methods and Applications*. Springer, London, 2006. ISBN 978-1-84628-029-0.

[56] Robin R Vallacher and Daniel M Wegner. What Do People Think They're Doing? Action Identification and Human Behavior. *Psychological Review*, 94(1):3–15, 1987.

[57] M. Kantardzic and IEEE Press. *Data mining: concepts, models, methods, and algorithms*. Wiley-Interscience, 2003. ISBN 0471228524. URL http://www.lavoisier.fr/livre/notice.asp?id=OKLWR3A3SA3OWF.

[58] Nong Ye. *The handbook of data mining*. Lawrence Erlbaum Associates, New Jersey, 2003. ISBN 0805840818. URL http://books.google.com/books?hl=en&lr=&id=vC3gdPDaf7IC&oi=fnd&pg=PR18&dq=The+handbook+of+data+mining&ots=SvD1obX1JX&sig=nod2L9tpPXWYK8QbD6JQ81pwI3M.

[59] D.J. Barrett, R.E. Silverman, and R.G. Byrnes. *SSH, the secure shell: the definitive guide*. O'Reilly Media, Inc., California, 2005. ISBN 0596008953. URL http://portal.acm.org/citation.cfm?id=1199540.

[60] IH Witten. Data Mining: Practical machine learning tools and techniques. *Machine Learning*, 2005. URL http://books.google.com/books?hl=en&lr=&id=QTnOcZJzlUoC&oi=fnd&pg=PP2&dq=Data+Mining:+Practical+machine+learning+tools+and+techniques&ots=3ghC9sTjRa&sig=ZtU3Ksg3EbJPuVespLWrMmek7UM.

[61] Hispasec. Virus Total. URL http://www.virustotal.com/es/.

[62] Vinod Yegneswaran, Hassen Saidi, Phillip Porras, and Monirul Sharif. Eureka: A framework for enabling static analysis on malware. In *ESORICS '08 Proceedings of the 13th European Symposium on Research in Computer Security*, number April, pages 481 – 500, Berlin, 2008. Springer-Verlag. URL http://dx.doi.org/10.1007/978-3-540-88313-5_31.

[63] Microsoft Security Intelligence. Microsoft Security Intelligence Report. Technical Report June, Microsoft, 2010. URL http://www.microsoft.com/security/sir/default.aspx.

[64] Mahbod Tavallaee, Natalia Stakhanova, and Ali Akbar Ghorbani. Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(5):516–524, September 2010. ISSN 1094-6977. doi: 10.1109/TSMCC.2010.2048428. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5464348.

[65] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, November 2013. ISSN 01674048. doi: 10.1016/j.cose.2013.04.007. URL http://linkinghub.elsevier.com/retrieve/pii/S0167404813000837.

[66] Peng Li, Limin Liu, Debin Gao, and Michael K Reiter. On Challenges in Evaluating Malware Clustering. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, pages 238—-255. Springer-Verlag, 2010. URL http://dl.acm.org/citation.cfm?id=1894166.1894183.

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

[68] Bernardo A. Huberman. Sociology of science: Big data deserve a bigger audience. *Nature*, 482(7385):308–308, 2012. doi: http://dx.doi.org/10.1038/482308d.

[69] M Rehak, M Pechoucek, Martin Grill, and Jan Stiborek. Adaptive multiagent system for network traffic monitoring. *Intelligent Systems, IEEE*, 24(3):16–25, 2009. ISSN 1541-1672. doi: 10.1109/MIS.2009.42. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4983378.

[70] J. Francois, Shaonan Wang, Walter Bronzi, Radu State, and Thomas Engel. BotCloud : Detecting Botnets Using MapReduce. In *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*, pages 1–6, Tenerife, Spain, 2011. ISBN 9781457710193. doi: 10.1109/WIFS.2011.6123125.

[71] Ricardo Baeza-Yates and Berthire Ribeiro-Neto. *Modern information retrieval*. ACM Press / Addison-Wesley, 1999. ISBN 0-201-39829-X.

[72] Jonathan J. Davis and Andrew J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security*, 30(6-7):353–375, September 2011. ISSN 01674048. doi: 10.1016/j.cose.2011.05.008. URL http://linkinghub.elsevier.com/retrieve/pii/S0167404811000691.

[73] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *SEC'11 Proceedings of the 20th USENIX conference on Security*, pages 29–29, 2011. URL http://static.usenix.org/legacy/events/sec11/tech/full_papers/Jacob.pdf.

[74] Tao Wang and S.Z. Yu. Centralized Botnet Detection by Traffic Aggregation. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 86–93. IEEE, August 2009. ISBN 978-0-7695-3747-4. doi: 10.1109/ISPA.2009.74. URL http://www.computer.org/portal/web/csdl/doi/10.1109/ISPA.2009.74.

[75] V. Jacobson, C. Leres, and S. McCanne. tcpdump. http://www.tcpdump.org, 1994. URL www.tcpdump.org.

[76] Shawn Ostermann. Tcptrace, 2009. URL www.tcptrace.org.

[77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. 39(1):1–38, 1997.

[78] Aiyou Chen, L Li, and Jin Cao. Tracking cardinality distributions in network traffic. In *INFOCOM 2009, IEEE*, pages 819–827, 2009. doi: 10.1109/INFCOM.2009.5061991. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5061991.

[79] M M Masud, Jing Gao, L Khan, Jiawei Han, and B Thuraisingham. A Practical Approach to Classify Evolving Data Streams: Training with Limited Amount of Labeled Data. In *Conference on Data Mining, 2008. ICDM '08. Eighth IEEE International*, pages 929–934, 2008. doi: 10.1109/ICDM.2008.152.

[80] Like Zhang. *A sublexical unit based hash model approach for spam detection*. PhD thesis, The University of Texas at San Antonio, 2009. URL http://dl.acm.org/citation.cfm?id=1751034.

[81] A McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In Ian Barakat, Chadi and Pratt, editor, *Passive and Active Network Measurement*, pages 205–214. Springer Berlin / Heidelberg, 2004. ISBN 978-3-540-21492-2. doi: 10.1007/978-3-540-24668-8_21. URL http://dx.doi.org/10.1007/978-3-540-24668-8_21.

[82] Mark Hall, Eibe Frank, and Geoffrey Holmes. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11:9, 2009. URL http://dl.acm.org/citation.cfm?id=1656278.

[83] S Yen and Y Lee. Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications*, 36(3):5718–5727, April 2009. ISSN 09574174. doi: 10.1016/j.eswa.2008.06.108. URL http://linkinghub.elsevier.com/retrieve/pii/S0957417408003527.

[84] Andreas Hegna. *Visualizing Spatial and Temporal Dynamics of a Class of IRC-Based Botnets*. PhD thesis, Norwegian University of Science and Technology, Faculty of Information Technology Mathematics and Electrical Engineering, 2010. URL http://ntnu.diva-portal.org/smash/record.jsf?pid=diva2:353050.

[85] B. AsSadhan, J.M.F Moura, and David Lapsley. Periodic Behavior in Botnet Command and Control Channels Traffic. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–6. IEEE, 2009. doi: 10.1109/GLOCOM.2009.5426172. URL http://repository.ksu.edu.sa/jspui/handle/123456789/12757.

[86] K Lee, J Kim, K Kwon, Y Han, and S Kim. DDoS attack detection method using cluster analysis. *Expert Systems with Applications*, 34(3):1659–1665, April 2008. ISSN 09574174. doi: 10.1016/j.eswa.2007.01.040. URL http://linkinghub.elsevier.com/retrieve/pii/S0957417407000395.

[87] T.K. Moon. The expectation-maximization algorithm. Technical Report 6, 1996. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=543975.

[88] Jan Jusko and Martin Rehak. Identifying Peer-to-Peer Communities in the Network by Connection Graph Analysis. *International Journal Of Network Management*, 24(4):235–252, 2014. doi: 10.1002/nem.

[89] Sebastian Garcia. Malware Capture Facility Project, 2013. URL https://mcfp.felk.cvut.cz/.

[90] Argus. Argus. Auditing Network Activity, 2013. URL http://qosient.com/argus/.

[91] R Lyda and J Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, 5(2):40 – 45, 2007. doi: 10.1109/MSP.2007.48. URL http://virii.es/U/UsingEntropyAnalysistoFindEncryptedandPackedMalware.pdf.

[92] Sebastian Garcia, Vojtech Uhlir, and Martin Rehak. Identifying and Modeling Botnet C&C Behaviors. In ACM, editor, *Proceedings of the 1st International Workshop on Agents and CyberSecurity (ACySE '14)*, New York, 2014. ACM. doi: 10.1145/2602945.2602949. URL http://dl.acm.org/citation.cfm?id=2602949.

[93] Cisco. NetFlow, 2009. URL http://www.cisco.com/en/US/products/ps6645/products_ios_protocol_option_home.html.

[94] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information, 2008. URL http://tools.ietf.org/html/rfc5101.

[95] Lorenzo Cavallaro, Christopher Kruegel, Giovanni Vigna, and F Yu. Mining the network behavior of bots. Technical report, UCSB, 2009. URL http://www.cs.ucsb.edu/sites/www.cs.ucsb.edu/files/docs/reports/2009-12.pdf.

[96] Ping Wang and Baber Aslam. Peer-to-Peer Botnets: The Next Generation of Botnet Attacks. *Electrical Engineering*, pages 1–25, 2010. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.6675&rep=rep1&type=pdf.

[97] John G Kemeny and James Laurie Snell. *Finite markov chains*. van Nostrand Princeton, NJ, 1960.

[98] PM Long and RA Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78(3):287–304, 2010. URL http://link.springer.com/article/10.1007/s10994-009-5165-z.

[99] Anton Dries and Ulrich Ruckert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2:311–327, 2009. ISSN 19321872. doi: 10.1002/sam.10054.

[100] Adam J. Aviv and Andread Haeberlen. Challenges in experimenting with botnet detection systems. In *USENIX 4th CSET Workshop, San Francisco, CA*, pages 6–6. University of Pennsylvania, 2011. URL http://static.usenix.org/event/cset11/tech/final_files/Aviv.pdf.

[101] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). Technical report, NIST, 2007.

[102] R.R Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183 – 190, 1988. doi: 10.1109/21.87068.

[103] L Ertoz, Eric Eilertson, Aleksandar Lazarevic, Pang-ning Tan, Vipin Kumar, Jaideep Srivastava, and Paul Dokas. Minds-minnesota intrusion detection system. In *Next Generation Data Mining*, pages 199–218. MIT Press, 2004. URL http://www.it.iitb.ac.in/~deepak/deepak/courses/mtp/papers/minds-minnesotaintrusiondetectionsystem.pdf.

[104] Kuai Xu, ZL Zhang, and Supratik Bhattacharyya. Reducing unwanted traffic in a backbone network. In *SRUTI 05: Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 9–15, 2005. URL http://static.usenix.org/event/sruti05/tech/talks/xu.pdf.

[105] K Xu and ZL Zhang. Profiling internet backbone traffic: behavior models and applications. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 169–180, 2005. ISBN 1595930094. URL http://dl.acm.org/citation.cfm?id=1080112.

[106] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM Computer Communication Review*, 34 (4):219–230, October 2004. ISSN 01464833. doi: 10.1145/1030194.1015492. URL http://dl.acm.org/citation.cfm?id=1015492.

[107] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *ACM SIGCOMM Computer Communication Review*, 35 (4):217–228, 2005. doi: 10.1145/1080091.1080118. URL http://portal.acm.org/citation.cfm?doid=1080091.1080118.

[108] Avinash Sridharan, Tao Ye, and Supratik Bhattacharyya. Connectionless port scan detection on the backbone. In *25th IEEE International Conference on Performance, Computing, and Communications. IPCCC*, pages 10 pp.–576. Ieee, 2006. ISBN 1-4244-0198-4. doi: 10.1109/.2006.1629454. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1629454.

[109] Tomas Pevny, Martin Rehak, and Martin Grill. Identifying suspicious users in corporate networks. In *Proceedings of workshop on information forensics and security*, pages 1–6, 2012. URL http://exile.felk.cvut.cz/pevnak/pdfs/Pev12-WIFS.pdf.

[110] Sarvapali D Ramchurn, Nicholas R Jennings, Carles Sierra, and Lluis Godo. Devising A Trust Model For Multi-Agent Interactions Using Confidence and Reputation. *International Journal of Applied Artificial Intelligence*, 18:833–852, 2004.

[111] Jordi Sabater and Carles Sierra. Review on Computational Trust and Reputation Models. *Artificial Intelligence Review*, 24(1):33–60, September 2005. ISSN 0269-2821. doi: 10.1007/s10462-004-0041-5. URL http://www.springerlink.com/index/10.1007/s10462-004-0041-5.

[112] Achim Rettinger, Matthias Nickles, and Volker Tresp. Learning Initial Trust Among Interacting Agents. In *Proceedings of the 11th international workshop on Cooperative Information Agents XI*, pages 313—-327. Springer-Verlag, 2007. doi: 10.1007/978-3-540-75119-9_22.

[113] Martin Rehak, Michal Pechoucek, Milos Gregor, and Martin Reh. Trust Modeling with Context Representation and Generalized Identities. In *Proceedings of the 11th international workshop on Cooperative Information Agents XI*, pages 298—-312. Springer-Verlag, 2007. doi: 10.1007/978-3-540-75119-9_21.

[114] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Stealthy Poisoning Attacks on PCA-based Anomaly Detectors. *SIGMETRICS Performance Evaluation Review*, 37(2):73–74, 2009. doi: 10.1145/1639562.1639592. URL http://dl.acm.org/citation.cfm?id=1639592.

[115] Luis Corrons. Mariposa botnet, 2010. URL http://pandalabs.pandasecurity.com/mariposa-botnet/.

[116] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, 1999. URL http://static.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf.

[117] L. Chappell and G. Combs. *Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide*. Laura Chappell University, 2010. ISBN 9781893939998. URL http://books.google.com.ar/books?id=MUAPRQAACAAJ.

[118] Anna Sperotto, Ramin Sadre, Frank Van Vliet, and Aiko Pras. A labeled data set for flow-based intrusion detection. In *IPOM '09 Proceedings of the 9th*

IEEE International Workshop on IP Operations and Management, pages 39–50, 2009. doi: 10.1007/978-3-642-04968-2_4. URL http://www.springerlink.com/index/b2g6274770j2752k.pdf.

[119] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 2010 ACM Conference on Emerging Networking Experiments and Technology, CoNEXT 2010, 2010*, pages 8–8, 2010. doi: 10.1145/1921168.1921179. URL http://dl.acm.org/citation.cfm?id=1921168.1921179.

[120] Sotiris Kotsiantis, Dimitris Kanellopoulos, and Panayiotis Pintelas. Handling imbalanced datasets : A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.

[121] Alexander K Seewald and Wilfried N Gansterer. On the detection and identification of botnets. *Computers & Security*, 29(1):45–58, 2010. ISSN 0167-4048. doi: DOI:10.1016/j.cose.2009.07.007. URL http://www.sciencedirect.com/science/article/B6V8G-4WY6JSP-1/2/22cf815a5f2205920fd11f6100899df0.

[122] SL Garfinkel and Michael Shick. Passive TCP Reconstruction and Forensic Analysis with tcpflow. Technical report, Naval Postgraduate School, 2013. URL http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA585499.

[123] Rian Shelly and Jonathan Valiente. oip, 2012. URL https://github.com/USU-Security/oip.

[124] Thomas Chopitea. When the hunter becomes the hunted. By Caveman. Technical report, 2013.

[125] Alberto Dainotti, Alistair King, Ferdinando Papale, and A Pescapè. Analysis of a/0 stealth scan from a botnet. In *IMC '12 Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 1–4. ACM New York, 2012. ISBN 9781450317054. doi: http://dx.doi.org/10.1145/2398776.2398778. URL http://dl.acm.org/citation.cfm?id=2398778.

[126] G Szabo, D Orincsay, Szabolcs Malomsoky, and I Szabo. On the validation of traffic classification algorithms. In *9th International Conference, Passive and Active Network Measurement, (PAM 2008)*, pages 72–81, Cleveland, 2008. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-79232-1_8. URL http://link.springer.com/chapter/10.1007/978-3-540-79232-1_8.

[127] NexGinRC. Endpoint Worm Scan Dataset, 2013. URL http://nexginrc.org/Datasets/DatasetDetail.aspx?pageID=24.

[128] Kenjiro Cho, K Mitsuya, and A Kato. Traffic data repository at the WIDE project. In *ATEC '00 Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 51–52. USENIX Association, 2000. URL http://dl.acm.org/citation.cfm?id=1267775.

[129] Lillian Ablon, Martin C. Libicki, and Andrea A. Golay. Markets for Cybercrime Tools and Stolen Data. Technical report, RAND Corporation, 2014.

[130] James Andrew Lewis and Stewart Baker. The Economic Impact of Cybercrime and Cyber Espionage. (July):1–20, 2013. URL http://csis.org/publication/economic-impact-cybercrime-and-cyber-espionage.

[131] M Marquis-Boire. From Bahrain with love: FinFisher's spy kit exposed. Technical Report Vm, The Citizen Lab, Toronto, 2012. URL https://citizenlab.org/2012/07/from-bahrain-with-love-finfishers-spy-kit-exposed/.

[132] Satoshi Kondo and Naoshi Sato. Botnet Traffic Detection Techniques by C&C Session Classification Using SVM. In Springer Berlin / Heidelberg, editor, *Advances in Information and Computer Security*, volume 4752/2007 of *Lecture Notes in Computer Science*, pages 91–104. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-75650-7. doi: http://dx.doi.org/10.1007/978-3-540-75651-4.