Botnet detection using correlated anomalies

Naveen Davis

Kongens Lyngby 2012
IMM-M.Sc.-2012-50

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2012-50

Summary (English)

Botnets are collections of computers which have come under the control of a malicious person or organization, and can be ordered to perform various malicious tasks such as sending spam mail,performing click fraud, farming personal or other confidential information, or performing distributed denial of service attacks. They are currently regarded as one of the major threats to the widespread use of the Internet, and finding ways to counter them is a challenge of great importance.

The goal of the thesis is to produce a simple prototype which detects botnet attacks by correlating patterns of anomalous behavior which develop in similar ways in different parts of a network, such as within a sub-set of the computers within a given subnet. In order to accomplish this we carried out a study of the literature on analysis methods of this type and decided to exploit a method which combines both host-level and network-level information to detect anomalous behavior. We selected a suitable platform and operating system to perform the analysis. We were able to obtain some valuable results from the analysis, but it was not enough to come up with a precise conclusion.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. in Security and Mobile Computing.

The thesis deals with a botnet detection technique by combing network-level and host-level information to find anomalous communication patterns. It also deals with machine learning techniques and algorithms used for training botnet detection systems.

The thesis consists of an introduction to the characteristics of botnets and the various detection techniques employed to defend against them. The subsequent chapters in this thesis discusses how correlation can be applied to combine the analysis results obtained from the network and the host system to accomplish a better detection result. The final chapters explains the design of our simple prototype used to perform the analysis and the important findings and challenges we faced during the analysis. Finally a conclusion is provided summarizing the lessons we learned and how we could possibly improve this technique in the future.

Lyngby, 30-June-2012

Naveen Davis

Acknowledgements

I would like to thank my supervisor Robin Sharp for our weekly conversations and for all the guidance and advices he has provided during the thesis project. I have learned a lot about various aspects of botnet detection challenges from our weekly converstions which helped me immesly in carrying out the thesis work.

I would also like to than my supervisor Professor Antii from Aalto university, Finland for all the valuable comments and feedback regarding my thesis project.

Contents

Chapter 1

Introduction

Malicious software commonly known as malware is defined as a program with malicious intent that has the potential to harm the machine on which it executes or the network over which it communicates [29]. Malware exits in different forms, but the most significant one at present is botnet. In the context of information security, a bot can be defined as a malware instance that runs automatically and autonomously on a compromised machine without the consent of the user[13]. Botnets are collections of computers which have come under the control of a malicious person or organization, and can be ordered to perform various malicious tasks.

The basic difference between botnet and other forms of malware is the existence of command and control (C&C) architecture. The one who takes control of the botnets is commonly knows as the Botmaster. The Botmaster communicates with the botnets through a command channel. Botnets were initially used as a way to automate tasks such as internet relay chats, online gaming etc. But with the technological advancements in networking, especially the internet, malicious users began to use botnets to automate illegal tasks such as sending spam mail, performing click fraud, farming personal or other confidential information, or performing distributed denial of service(DDoS) attacks.

Botnets are currently regarded as one of the major threats to the widespread use of the Internet. Over the past decade botnets are heavily used for all kinds of computer crimes such as phishing, distributing pirated media and software, identity theft, adware, stealing information and computer resource and so on [36]. Majority of these attacks are focused on making money through illegal means. Hence finding ways to counter botnets is a challenge of great importance.

Detecting botnet at an early stage is crucial since they are reusable and renewable resources. Botnet detection faces a number of challenges [2]. Botnet controllers have access to all kind of real world data through illegal means, whereas botnet researcher is often left with a limited amount of data due to administrative and privacy regulations. Another challenge is that the botnet controllers are making use of the latest techniques and covert ways to control their bot. Heterogeneity of the internet is the next challenge as it involves networks with different characteristics. Overall botnet detection still remains one of the most challenging research area in the field of internet security.

1.1 Problem statement

A large number of methods have been proposed in the literature for detection and tracking of the botnets. Majority of this methods can be classified in to two approaches as specified in [41]. The first approach is to detect botnets by setting up honeynets (network of computers, usually protected by a firewall to regulate traffic). The other approach is based on intrusion detection systems (IDS) which has been further categorized in to signature based and anomaly based detection techniques.

The focus of this thesis project is based on IDS approach. As the first step of botnet detection we need to get a clear understanding of the behavior of botnet malware on an infected host. Several techniques exists, but our aim is to exploit a method which detects botnet attacks by identifying patterns of anomalous behavior which develop in similar ways in different parts of a network, such as within a sub-set of the computers within a given sub-net. One of such methods where detection of botnets based on network behavior is explained in [35]. The method makes use of network flow characteristics such as bandwidth, packet timing and burst duration for evidence of botnet command and control activity. In addition to this , if we can correlate the network behavior to system call events, the detection technique could be more efficient.

Now a number of research question arises. Are these features adequate for efficient botnet detection? Should we also look into other network flow characteristics? What is the volume of network traffic required to reach a reasonable conclusion? The number of false positives and false negatives? Overall time taken for botnet detection?, How can we co-relate network behavior and in-host events and What are the performance penalties?

In order to find possible answers for this research questions we have to study the literature on existing botnet detection methods. Once done with the literature analysis, we need to analyze the network and host level activities on a suitable platform and operating system. In order to accomplish this a test environment need to be set up. The final aim of this thesis is to design and implement a prototype based on our test results. The analysis performed by this prototype, hopefully would provide the answers to our research questions.

1.2 Thesis Outline

The structure of the thesis report is explained briefly as follows. Chapter 2 provides an in depth discussion of the the two common approaches for botnet detection, namely honeynets and IDS. Section 2.1 describes the botnet characteristics by explaining the common and advanced features. The two main detection techniques are explained in detail in section 2.2 along with its advantages and disadvantages. Both these techniques can be applied either at the host-level or at the network-level. Section 2.3 described how a combination of host-based and network-based techniques could be made use to create a more powerful botnet detection mechanism.

Correlation of anomalies is a powerful technique for detecting botnets. Chapter 3 explains how this technique is made use in the real world. The success of this technique depends on the extent which we can analyze the behavior of a botnet infected network. Section 3.1 discusses about behavioral analysis. The main principles of behavioral analysis techniques and some of the recent research conducted using this technique is discussed with some test results. The key aspects of botnet data description: feature extraction and classification are explained in section 3.2. In order to differentiate between the normal traffic and the attack traffic we need to know which features to be extracted from the network traffic data. To accomplish this we need to make use feature extraction techniques. Once we extract the required features, the next step is to classify these features based on some kind of pattern matching. We have chosen clustering technique based on unsupervised learning algorithm. The final section of this chapter discusses the evaluation criteria.

Chapter 4 represents our experimentation environment. In today's internet world, a botnet controller will have, at the touch of a button, thousands of compromised computers (bots) ready to execute the malicious commands issued by the controller [2]. On the other hand majority of the botnet researchers will not have this luxury. Hence they would require to simulate such an scenario to analyze data and extract useful information for detecting botnets. Hence we make use of virtual machines to simulate a small computer network. Another important reason for opting virtual machines is the fact that analyzing botnets on a real world system may have huge security implications. Also getting an approval for conducting such an experiment normally tends to take a huge amount of time, if at all possible. Section 4.1 describes this test setup. The tools used for analysis is discussed in section 4.2.

Chapter 5 depicts a simple prototype developed for analyzing host as well as network behavior to detect botnets. Section 5.1 explains in detail how the system is trained in order to differentiate between normal and anomalous traffic. This section also contains the detailed discussion about how the data set for training was generated and collected for analysis. In section 5.2 we depict the high level system design of our prototype explaining the architecture of the co-relation engine. Detailed description of the individual components and the implementation strategy is also explained in this section.

Our analysis is specified in Chapter 6. After setting up the small computer network using virtual machines, we have to filter out the network traffic that is irrelevant. Packet filters can be made use to accomplish this task. The next step is to analyze the network traffic which is discussed in section 6.1. Another important activity is to correlate the analysis results of network traffic with system events on the host, which is discussed in section 6.2. The final part of this chapter, section 6.3 discuss the various challenges faced and the lessons learned form our analysis.

The conclusion for the thesis report is provided in chapter 7. Section 7.1 discussed the possible future work with respect to our thesis.

Chapter 2

Botnet Detection

Botnets are collections of computers which have come under the control of a malicious person or organization, and can be ordered to perform various malicious tasks. The highlight of botnets as a phenomenon is fact that they can provide anonymity through a multi-tier command and control architecture[36]. The person who orders the commands is often referred to as botmaster. Botnet detection is becoming more and more significant because of the fact that at present they are one of the major threats on the web, conducting subtle attacks using large coordinated groups of hosts. The scale of cyber crime attacks on the internet by botnets are increasing day by day. Numerous research are being carried all over the world to combat botnets. On the other hand botnets are evolving rapidly making it even more harder to detect and defend against. In this chapter we discuss about botnet detection techniques. To start with we provide a detailed discussion about botnets and their characteristics in general. The subsequent section discusses the two main approaches taken by the researchers in order to detect botnets namely, honey nets and intrusion detection systems (IDS)[41].

2.1 Botnet Characteristics

The most important characteristic of a botnet is its communication structure which is used to command and control (C&C) the infected hosts on a network[30]. The botmaster controls the C&C server who stores the information about infected hosts. In addition to that it holds the full list of malicious commands that needs to be send across to the bots. The typical life cycle of a bot can be explained with the help of Figure 2.1 [36].

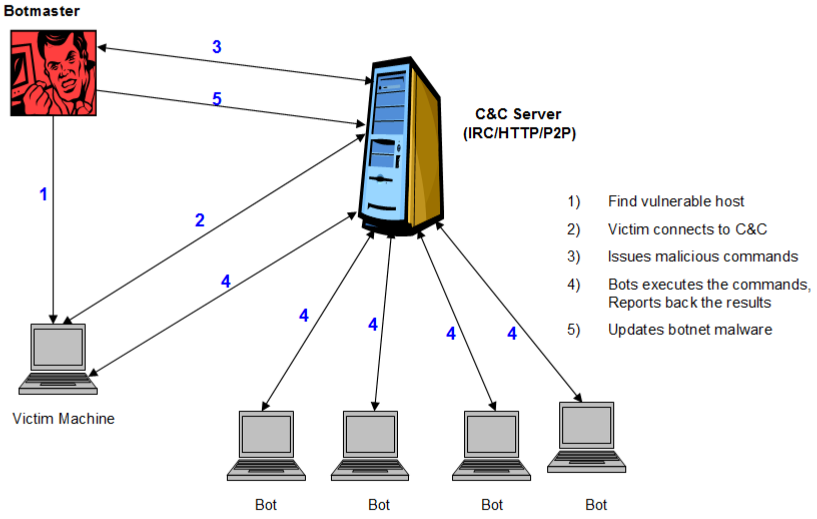

Botmaster
3
5
C&C Server
(IRC/HTTP/P2P)
1
2
4
4
4
4
4
1) Find vulnerable host
2) Victim connects to C&C
3) Issues malicious commands
4) Bots executes the commands, Reports back the results
5) Updates botnet malware
Victim Machine
Bot
Bot
Bot
Bot

Figure 2.1: Basic botnet life-cycle

As the first step the botmaster needs to find a victim to exploit. To accomplish this the botmaster scans through the infected network to find security vulnerabilities on each host in the network. Once it finds the target victim it infects that host by exploiting the required security vulnerabilities. In the second step the infected bot connects to the C&C server and listens on the command channel to receive the orders from botmaster. The C&C server records the information about the new victim along with the list of security vulnerabilities of the compromised host. As a third step the botmaster sends the malicious commands to all the bots through C&C server and the C&C server reports back the obtained results to the botmaster. In step 4 the bots downloads these malicious commands and executes it as required,and the cycle continues. The final step in the cycle is executed when the botnet code needs to be updated. This is done by the botmaster through an update command issued to C&C server to install new version of malware on the bots.

The type of command and control communication between the bots or between the bot and the bot master can be differentiated in to two types namely, push-based commanding or pull-based commanding [14]. In push-based control communication, as the name suggests, the botmaster pushes the commands that needs to be executed to the bots. This means that in this type of control communication the botmaster has real time control over the bots. The use of internet relay chat (IRC) servers for command and control is a typical example for push-based communication. On the other hand in pull-based communication, the botmaster stores the commands to be executed in a file and the bots are allowed to fetch this commands from the file in a periodic manner. An example for pull-based communication is the use of HTTP protocol for command and control which in turn is used for spamming.

2.1.1 Common Features

The command communication between botnets can be based on protocols such as IRC, HTTP, DNS or P2P[44]. If we look at the evolution of botnets, in the beginning, most of them used a centralized approach for managing the bots. This was mainly accomplished using various versions of IRC protocol. According to[32] IRC was the popular choice because of the following reasons; a) the interactive nature of client-server communication in IRC, b) availability of the source code readily for easy modifications, c) ability to control multiple botnets using nicknames for bots, d) password protected channels and e) redundancy achieved by linking several servers together. IRC based bots are controlled via IRC channels through which the bots communicates to the botmaster. The botmaster will set up IRC server in order to issue malicious commands to the bots. When a new computer is compromised, it tries to contact the IRC server by using the information specifies in the bot program.

The second most popular command communication structure used by botnets are web servers (HTTP) [1]. A HTTP based bot connects to a web server that is controlled by botmaster, receives commands from it, performs the actions required and sends back the response to the web server. HTTP was chosen by the botnets for the reason that most firewalls cannot distinguish between a web-based bot traffic and legitimate web traffic. Instead of connecting to C&C web server specified in the botnet code, a bot can make use of domain name (DNS name) in order to avoid black-listing of the IP address of the web server or shutting down the web server. Thus if a C&C web server is taken down or IP address of the server is blocked, the botmaster just needs to update the DNS mapping to point to a new C&C domain or assign a new IP address.

Based on the command and control architecture, botnets can be mainly classified into two architectures namely centralized and peer to peer (P2P) as shown in Figure 2.2[10]. In the centralized architecture the botmaster controls the C&C server. The C&C server in turn communicates with all bot agents and sends them the malicious commands given by botmaster. The bots executes these malicious commands and when a new victim is compromised the bot reports back to the C&C server. The C&C server registers the new victim and the cycle continues. IRC, HTTP and DNS based botnets are all examples for centralized architecture. The main drawback of centralized architecture is the fact that if the C&C server is taken down the botnet will be eliminated. In other words all the botnets based on centralized architecture are susceptible to single point of failure. To avoid this the botnet community decided to shift the architecture from centralized to peer-to-peer (P2P).

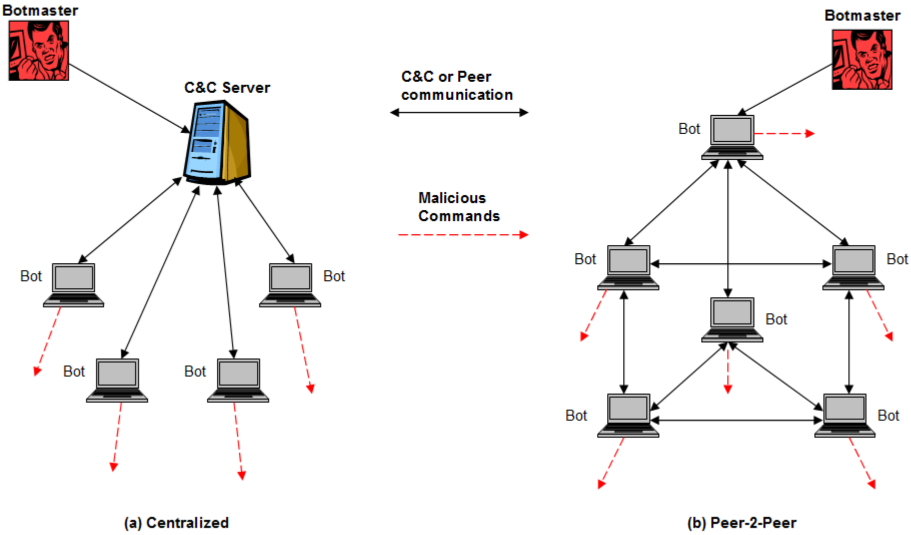


Figure 2.2: Two main botnet architectures

In P2P architecture there is no centralized server. In fact all the bots in the network act as both a bot server and client. Hence the network of botnets continues to function even if one of the bot server is eliminated. The botmaster shares the command over the P2P network in order to issue commands to the bots. In addition to that the botmaster published specific search keys that can be used by the bots to find the shared command files. The bots communicates with each other transferring the command files and search key specified by the botmaster so that they can locate and execute the related command. P2P bots are still not popular as the other bots mainly because of the complicated nature of C&C server in the P2P architecture. Also the fact that it is easy to detect other infected peers if we could track down one of the peer bot [1] makes it unpopular.

2.1.2 Advanced Features

Botnets are becoming more and more sophisticated day by day. The botnet creators are making use of many advanced features in order to obfuscate their malware against detection [3]. In this section we will discuss briefly some of the main techniques that are used to make it difficult for botnet defenders to detect and analyze the bot malware. The basic botnet detection mechanism is accomplished by signature matching. The simplest form of obfuscation technique for botnet malware is dead code insertion. But modern signature detectors would easily detect the variations created by dead code. A powerful technique that could be used by the bot designers to evade signature detection is polymorphism. Botnet malware that incorporates polymorphic techniques used random encoding to evade signature matching. Another technique to thwart signature detection includes the use of packing followed by encryption of bot binaries. This means that the signature of bot binaries changes and hence would go undetected by botnet signature detectors.

Modern botnets make use of fast-flux techniques in order to increase availability of C&C servers by hiding the actual severs responsible for the updated copies of the malware [1]. The basic idea behind this technique is to change the DNS to IP mapping of the download location of the bot malware constantly. This means that the botmaster can still control the C&C server since blocking an specific IP address does not help us in this case. Botnets also make use of deception techniques to evade detection, for e.g. rootkits. Botnet detection in the research community is often carried out on virtual machines due to privacy and security reasons. There are a few botnet applications with the capability to check if the infected host is running in a virtual machine, for e.g[8] describes an approach where they make use of a timing based approach to detect virtual machine monitors without relying on the implementation details of virtual machine. As a result botnet may become inactive or destroy itself, thereby defeating detection mechanisms leaving researches with any meaningful results.

2.2 Detection Techniques

Botnet detection, at present, is a hot topic in the research community. In recent years a number of botnet malware have been collected and analyzed comprehensively by a number of researchers in this filed. These results were in turn used to detect botnets by developing malware signatures in anti-virus software. For successful botnet detection we need to look at the key characteristics related to bot malware. Monitoring abnormal host level activities can be used for detecting botnets. According to [32]the majority of the common characteristic of a bot malware are related to network activities since the bots require some sort of interaction with the command and control servers. Some of the common activities one could monitor to detect botnets are, opening of specific ports, establishing a number of unwanted network connections, downloading and executing files and programs, creating new processes with well known names, disabling anti-virus software and so on. The botnet detection techniques can be mainly classified in to two; Honeypots and Intrusion Detection Systems (IDS)[41].

2.2.1 Honeypots

A honeypot in the context of botnet detection can be defined as a computer system that is closely monitored for potential infiltration [34]. A honeypot is made vulnerable to malicious attacks deliberately so that the botnets can easily compromise the system. Once these systems are compromised we could try to extract various information about the botnet by monitoring all the activities. Honeynets refers to a collection of honeypots. The main goals of honeypot based botnet detection methods is to understand the types of attack vectors used by the botnet in operating system as well as the exploit code corresponding to these attack vectors. In addition to that we could analyze the software used for attacks and the actions performed by this software on the infected machine. Moreover we could investigate the malware binaries in detail that could then used for the detection of botnets.

A honeypot can take the role of either a client or server. A client honeypot emulates a normal user and looks for malicious servers to get attacked whereas a server honeypot emulates a legitimate service and stay passive while waiting to be attacked. According to[34] honeypots can be characterized by degree of interaction when responding to actions. High interaction honeypots are mainly used for serious research to track down different kinds of botnets. They typically collect up huge amounts of data by infiltrating itself through compromise. Since this compromised machine can be used by the botnet for further malicious activities, we need to put in place stringent measures to avoid any further damage to other systems in our network. This could be really hard work and can often end up in a total compromise of the network. High interaction honeypots usually have a number of hidden components in order to monitor the attack activities of the botnet. Honeynets consisting of high interaction honeypots are made use in order to obtain more fine grained details about the botnet, say for e.g. two honeypots could run different operating systems so that we could analyze the difference in botnet behavior with respect to the underlying operating system.

Low interaction honeypots are mainly used for emulating certain vulnerable programs to look real. It means a low interaction honeypot can separate the underlying system from the targeted system presented to the attacker. This could be beneficial for analyzing the botnet directly on the honeypot since the system on which it is running remains trusted. But on the other side since only a few vulnerable programs are emulated, only the botnet code that used these programs could be analyzed. Also a clever botnet could easily detect these kinds of simulations.

In general honeypots plays an important role in detecting and analyzing botnets[34]. They can be used to analyze malware code to generate anti-virus signatures. From the above discussion it is clear that in order to obtain maximum benefit from a honeypot, it need to classify network traffic and internal activities clearly. This is not a trivial task, and requires a detailed and close monitoring of all the processes which requires a huge amount of time. The main drawback with honeypot based method is the fact that it can only see the incoming traffic from the IP address assigned to it, which may not be enough for a comprehensive analysis and detection of a botnet.

2.2.2 Intrusion Detection Systems

The main goal of an intrusion detection system (IDS) is to detect intrusions, which are often security violations, by gathering and analyzing information from different sources within the network [17]. These intrusions are detected by comparing the ongoing evidence of intrusions with the known intrusion patterns known as signatures. Hence every IDS needs to have huge database of signatures in order to detect possible intrusions. If the pattern deviates from the signature specified the IDS alerts the system administrator about the suspicious activity. Moreover with appropriate rules any further attempts for similar compromise can be dealt by the IDS itself. In the context of botnet detection IDS can be classified into two, signature based detection and anomaly based detection[41].

2.2.2.1 Signature based detection

Signature based detection in IDS is accomplished with the available knowledge of useful signatures of existing botnets[39]. This technique is also known as misuse detection. This techniques requires that the IDS should have an up to date collection of all the possible bot signatures for successful detection. The advantage of this technique is the fact that the botnet can be detected immediately and there wont be any false positives at all. But keeping track of an updated database of all botnet signatures is not practical. This means if we don't have the botnet signature in the database the bot may go undetected. This is high likely since new bots are evolving on a daily basis. Also clever bots may make use of slightly different signature by obfuscation techniques, for e.g. changing their attack pattern by inserting dead code, thereby going undetected by the signature based IDS.

Signature based detection can be done either at the host level or at the network level[41]. If performed at the host level the signature would reflect activities and attempts to access operating system i.e. system calls, file and disk operations. Anti-virus software is a typical example of signature based detection at host level. A limitation of this detection technique is that host-based systems are at the same privilege level as bots on the same host, meaning the bot can disable anti-virus software and even use rootkit techniques to protect themselves from detection. On the other hand if performed at the network level the signatures would reflect activities of network protocols and features. They will be monitoring the entire network instead of individual systems. Many network security tools have made use of this signature based detection to good effect. For e.g. snort[31] is an open source IDS that monitors network traffic to detects intrusions by comparing the patterns with predefined set of rules and signatures. But as discussed earlier the drawback of the tool is that the rule set and signature should exist in order for the detection of the botnet. To sum up signature based detection will not work with unknown bots or zero-day bot attacks.

2.2.2.2 Anomaly based detection

Anomaly based detection defers from signature based detection due to the fact that in anomaly based technique the detection is based on deviation from normal behavior[39]. The profiles or templates of normal behavior and activities needs to be generated and stored in the IDS in order to apply anomaly based detection. All the activities that deviates from the normal behavior will be detected my IDS as suspicious. This will in turn generated a number of false positives. But on the positive side this technique is far better that signature based detection since it can detect new malicious activities for which the attack signature is not known. Anomaly based detection also can be done either at the host level or at the network level[41]. When applied to host level the detection is based on the system internals instead of network traffic. There are not many tools that implement this technique at host level. This is because it is hard to define normal behavior just by monitoring events at host level and it also suffers from the performance overhead caused by monitoring all invoked system calls.

Anomaly based detection at network level detects botnet based on number of network traffic anomalies that are not defined normal. Such network activities normally relates to high network latency, sudden burst in network traffic, traffic on unusual ports, a large number of failed connection attempts and so on. This means the detection technique will require a long time in order to find a deviation from a normal behavior pattern to detect possible botnets. A number of tools have adopted this technique successfully for botnet detection. For e.g the BotSniffer[13] tool detects the C&C channel of the botnet in a local area network. The tool accomplishes this detection based on the observation that the bots within the same botnet would demonstrate strong synchronization in their responses and activities. Hence the tool will be able to detect botnets without prior knowledge of signatures. One of the main limitation of this technique that if the botnet traffic is similar to normal traffic the bot may go undetected generating more false positives. This is highly possible since many botnets utilizes some regular protocols for C&C communications generating traffic similar to regular traffic.

2.3 Combining Host-based and Network-based Techniques

As discussed in previous sections the bots can be detected either at the host level or at network level. Both methods has its own advantages and disadvantages. Combing both the methods could facilitate a more powerful detection mechanism since it would provide a complete view of botnets behavior both at the host level as well as the network level. The basic idea behind this technique is that majority of the botnets requires a co-ordination between the network level and the host level for successful infection. This will lead to various kinds of malicious behaviors at both levels. In other words if multiple hosts behave similarly in the trigger-action patterns, they are highly like to be part of the same botnet. Hence these patterns can be grouped in to same suspicious clusters. Now since bots within the same botnet are likely to receive the same command requests from the botmaster a similar classification can be applied at network level with the help of IDS based detection techniques. As a final step a co-relation needs to be identified between the host level and network level behaviors based on suspicion levels.

Various research[33, 43, 12, 11, 9] have shown that correlating the malicious activities happening at host and network level provides a better detection rate. Most of these research performs passive detection. Hence our thesis work makes use of a these techniques as a basis in order to design a system that would detect anomalous behavior in real time. The next chapter discusses how anomalies are co-related based on botnet detection strategies and what key features of botnet, both at host and network level, are taken in to consideration to accomplish this.

Summary

In this chapter we provided a brief discussion about the typical characteristics of botnets and the common detection techniques . The important characteristics of botnets are explained along with their advanced features. Botnet detection techniques can be presented in different dimensions.We focus on the most common dimension which is whether the detection is based on the host (honeypot) or om the network (IDS). The chapter concludes with a brief discussion on how we can combine both host-based and network-based techniques to produce better botnet detection results.

Chapter 3

Correlation of Anomalies

To date botnet C&C communication pattern recognition for detection remains one of the most challenging tasks in the field of intrusion detection systems[39]. The main reason being the complexities involved in traffic classification with the evolution of Internet. Classifying attack traffic from normal traffic is an important basis of network security. Botnets are making use of such new techniques, disguising attack traffic with normal traffic , making it difficult to detect and defend against them. In order to achieve better results in detecting botnets, the anomalies need to be co-related. This is because of the fact that bots in a botnet generally behave in a similar manner, which is considered to be a reasonable assumption to make till now. So if we could find a relevant co-relation between bot activities it could be vital in terms of detecting them. Technically this is achieved by conducting a behavior analysis using machine learning techniques both at host level and network level.

Machine learning techniques are preferred for behavioral analysis because they do not require explicit signatures to classify malware programs. In fact the classification is done based on finding common features and co-relating different activities of the malware.

3.1 Behavioral Analysis

Botnets are often defined as a group of compromised computers (bots) that perform similar communication and malicious activity patterns within the same botnet[40]. This is the basic principle behind conducting behavioral analysis to detect botnets. Based on the nature of the botnet the attack activities varies from performing scanning of systems to emailing of spam and viruses, distributed denial of service (DDoS) traffic generation, accessing system files and resources and so on. Monitoring these behavioral activities will help us to detect and defend against botnets. The best possible result from these analysis could be the possible information regarding the C&C communication connections leading to the location of the botmaster. Behavioral analysis can be done either at the host level or at the network level. But as discussed in the previous chapter the combination of the results obtained from both the levels offers better detection rate. Guofei Gu proposed a co-relation based framework consisting of three different co-relation techniques for effective network based botnet detection in an enterprise-like environment such as a university campus network, or simply a local area network (LAN)[9].

The three different co-relation techniques based on botnet behaviors are vertical (dialog) co-relation, horizontal co-relation and cause-effect co-relation as shown in Figure 3.1[9]. Based on these co-relation techniques four detection systems have been developed namely, BotHunter, BotSniffer, BotMiner and BotProbe.

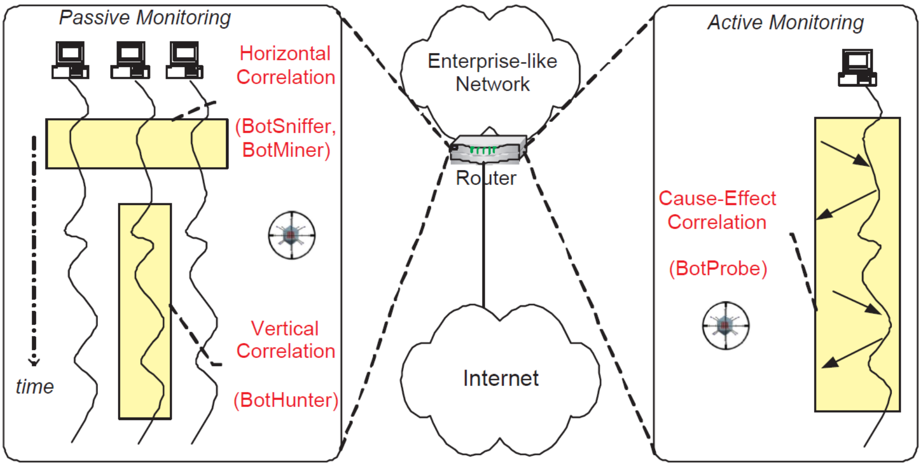


Figure 3.1: Different co-relation techniques for botnet detection[9].

BotHunter[12] detection system makes use of vertical co-relation to examine the behavioral history of each host in the network. Vertical co-relation is also known as dialog co-relation because of the fact that botnet detection is accomplished by recognizing co-related dialog trials between the internal assets and external entities across multiple stages. The collected evidence trail of data exchanges are matched against a state-based infection sequence model to detect botnets. As described in[12], BotHunter consists of a co-relation engine driven by several malware focused network detection sensors. Each of these senors are charged with detecting specific aspects and stages of the botnet infection process. These aspects and stages include inbound scanning, exploit usage, egg downloading, outbound bot coordination dialog and outbound attack or propagation. The co-relation engine of BotHunter links the dialog trail of inbound intrusion alarms with those outbound communication patterns indicating highly probable local host infection. If the evidence trail matches the already defined infection sequence model of BotHunter, a botnet detection report is generated capturing all the relevant events during the infection process. The main limitation of BotHunter tool is that it is restricted to predefined infection life cycle model[12].

BotSinffer[13] and Botminer[11] detection systems make use of horizontal co-relation to examine behavior similarity across multiple hosts. Both of these detection systems has an advantage over the other systems as they do not necessarily require to have botnet specific signatures or behavioral analysis of multiple different stages of an individual host. The basic idea behind horizontal co-relation is that bots within the same botnet will likely behave in a similar manner because of the already programmed activities related to the C&C server under the control of the botmaster. In most of the cases this behavior is not likely to be exhibited by normal hosts. The main aim behind the BotSniffer detection system is to figure out centralized C&C channels by monitoring the similarity between various spatial temporal co-relation activities[13]. On the other hand BotMiner presents a more general detection framework which is independent of botnet C&C control protocol and structure[11]. The basic working principle of BotMiner detection system is to cluster similar communication traffic and similar malicious traffic in the monitored network. Once clustering is done, botnet detection is attained by cross cluster co-relation to identify hosts that share both similar communication patterns and similar malicious activity patterns.

The co-relation techniques can be used for both, passive monitoring and active monitoring of botnet traffic as shown in figure 3.1. The botnet detection systems, BotHunter, BotSniffer and BotMiner uses passive monitoring technique. This usually requires relatively a long amount of time, because multiple stages or rounds are needed to observe botnet activities and to come up with a good detection result. This limitation can be overcome by making use of active monitoring strategy. BotProbe detection uses active monitoring strategy to reduce the time required for detection[9]. The key principle behind BotProbe detection system is the cause-effect co-relation caused by the command-response pattern between the bots and the C&C server. This detection system can actively participate in a network session even by injecting some well crafted packets to the monitored hosts if required. To obtain the best possible result, one could make use of combination of detection systems, those explained already in this section. These tools complement each other very well, facilitating better botnet detection in an enterprise-like network.

The focus of this thesis is to make use of the active monitoring strategy which is quite useful in real-world situation. The basic aim is to conduct a horizontal co-relation and a cause-effect co-relation in real time instead of passive monitoring. If the results obtained from both of these techniques could be related by identifying similar communication patterns, botnets could be detected in considerably short amount of time, compared to passive monitoring and obtaining better detection results. This could be quite complex and has a number of challenges which are explained in the rest of this report. But to start with we need to have a good set of data that clearly defines a botnet which is the focus of next section.

3.2 Botnet Data Description

In order to develop a successful botnet detection system we need to find key features that would be able to describe the botnet data. Once key features are selected the next step would be to apply a suitable classification algorithm on these features to group them based on similar communication patterns. These results is used to differentiate between normal and attack traffic. The subsequent sections discusses the key features which were selected and the details of the classification algorithm applied.

3.2.1 Feature Selection

There are a number of features which represents host as well as network activities. Not all of these features will be relevant for the botnet detection processes, for example classification task. We need to find out optimal features so that we can improve the performance of botnet detection processes, especially the classifiers. The fastest approach to select appropriate features is to make an independent analysis based on the data characteristics [19]. For botnet detection we need to look in to the important characteristics exhibited by the bot, both at host and network level to select appropriate features.

The bots share certain behavioral patterns at host level which are different from benign applications. According to[43], these behavioral activities can mainly be grouped into three categories taking place at the registry, file system and network stack. The basis for this classification is based on the fact that, typically a bot creates an exe or a dll file in the system directory as the first step in infecting a computer. This is followed by various file system activities in accordance with the instructions in the bot code. The final step is to open one or more ports to communicate with the botmaster via the C&C server. We should note that these activities may be carried out by a non-malicious process also. But it is highly unlikely that a non-malicious process will carry out these activities in a combined and aggregated manner. Therefore, we need to monitor system level activities of a host using appropriate tools which should include timestamp information. Information regarding various file operations, for e.g. read, write, carried out by the processes should also be recorded. Within the host system the network stack level information could be fetched from various intrinsic features of network traffic such as duration, service, source and destination address, source and destination ports, number of data bytes transferred, protocols and so on. This could be accomplished by using a network traffic analyzer tool that monitors all incoming and outgoing traffic of the host.

Identifying botnets based on network features is very complex task. Also it is hard to select relevant network features because of plethora of available features. If we do not select the appropriate features it would adversely effect the accuracy of the classification scheme. A good starting point to look for network traffic features for intrusion detection is specified in[21]. In[38], a model for investigating next generation botnets looks into eight key features of network traffic to characterize bot behavior. These network features are namely, total number of packets, packet size in bytes, number of different IP addresses contacted, number of different ports contacted, number of UDP, HTTP, SMTP packets and non ASCII bytes in payload. Botnet detection is carried out by identifying suspicious activities in the network patterns by monitoring these features for specified time intervals. Results shows a very good detection rate of bot traces and hence these features could be chosen for our analysis also. Once appropriate features that characterize botnets are selected, some amount of preprocessing should be done before applying classification.

3.2.2 Classification

Classifying network traffic into different applications is a very challenging task and still an active area of research. This is mainly because, we can no longer depend on port numbers to classify the network traffic. The emergence of P2P networking, tunneling applications, and other new protocols being the main reasons for these classification challenges[26]. One way to overcome these classification challenges is to examine payload signatures of the applications. But due to privacy concerns and the usage of encryption techniques, payload examination is not feasible for a large volume of network traffic. So in order to overcome these challenges, machine learning based methods are used in network traffic classification. The great strength of machine learning methods in network traffic classification is the ability to gain new knowledge or skills in a continuous fashion and to re-organize this knowledge to improve the classification performance[23].

Machine learning techniques can be divided into supervised and unsupervised approaches[19]. The supervised machine learning approach requires the training data to be labeled before the model is built. This is a difficult, error-prone as well as time consuming task. But the biggest limitation of this approach is that they wont be able to discover new applications, which is a key factor in botnet detection. Hence the supervised approaches are mainly used to improve the accuracy of classification processes. Unsupervised techniques on the other hand do not require pre-labeled data traces to build the model thus facilitating on-line learning and improving detection accuracy. They group the data traces based on information gathered from similarities and differences among the samples. Hence unsupervised technique can identify new applications by examining the flows that forms a new group. This make unsupervised machine learning technique ideal for detecting new botnet traces.

3.2.2.1 Clustering

Clustering is defined as the organization of data patterns into groups based on some measure of similarity[25]. In the context of botnet detection the most difficult hurdle facing clustering techniques is determining the number of clusters or groups. This is because, the occurrence of intrusions is unknown beforehand. The general approach for clustering techniques being used in botnet detection is to assume that the data traces are always divided in two categories namely, normal clusters and intrusive clusters. Also, the number of normal data traces largely outnumbers the number of intrusion traces.Clustering is an example for unsupervised learning algorithm which is widely used for network traffic classification[6]. There are a number of different unsupervised clustering algorithms, like K-Means, DBSCAN, etc, that can be employed for traffic classification. Clustering algorithms uses a distance measure between two feature vectors to group them into similar or different clusters which could be denoted by $dist(x, y)$.

Two types of data clustering algorithms are available, hierarchical clustering and partitional clustering[19]. In hierarchical clustering, the algorithm breaks up the data in to hierarchy of clusters, whereas, in partitional clustering the algorithm divides the data into mutually disjoint partitions. The main disadvantage of partitional clustering when compared to hierarchical clustering is that it need to know the number of clusters beforehand and the output results are strongly influenced by the choice of number of clusters. Hence, it would be necessary in most cases to apply the partitional clustering algorithm several times, with different number of clusters and evaluate the results to find the ideal number of clusters required, for the task in hand. But in terms of running time, partitional clustering is faster than hierarchical clustering which makes it a good candidate algorithm for network traffic classification.

A popular choice of partitional clustering algorithm used for traffic classification is the **K-Means** algorithm[6, 24]. There are several factors which contributes to the choice of K-Means algorithm when compared to other sophisticated clustering algorithms. The key reason being its simplicity and ease of implementation. This reduces the computational overhead of classifiers as the data structures representing the clusters allow fast computation of $dist(x, y)$. Another reason being the less amount of time required by K-Means algorithm to train the system compared to more complex clustering algorithms. Finally the classification accuracy achieved by the complex clustering algorithms compared to K-Means algorithm is negligible when taking in to account all other factors[24].

The K-Means algorithm can explained with help of a flow chart as shown in Figure 3.2[25]. The aim of the algorithm is to classify a given data set through a certain number of clusters fixed a priori. Let us assume that we have taken the initial number of clusters as k. The first step is to define$k$ centroids, one for each cluster. This is a key step because the output is dependent on the placement of centroids. Hence a better choice of placement would be to define these centroids far away from each other so that a clear distinction is made between each clusters. Second step of the algorithm is to take each point belonging to a given data set and calculate the distance to the centroids. Once the distance between the all data points and centroids are calculated, the third step is to group the data points in to the nearest centroids based on the result. The fourth step is to re-calculate the positions of new centroids based on the new grouping results and the cycle continues until the new grouping are same as the previous ones. Once the centroids do not move any more the K-Means clustering has reached its stability and no more iteration is needed. The algorithm returns the resulting no of the clusters containing the grouped data points. As discussed earlier the similarity between two data objects is computed using a distance function. We make use of the most commonly used distance function which is the Euclidean

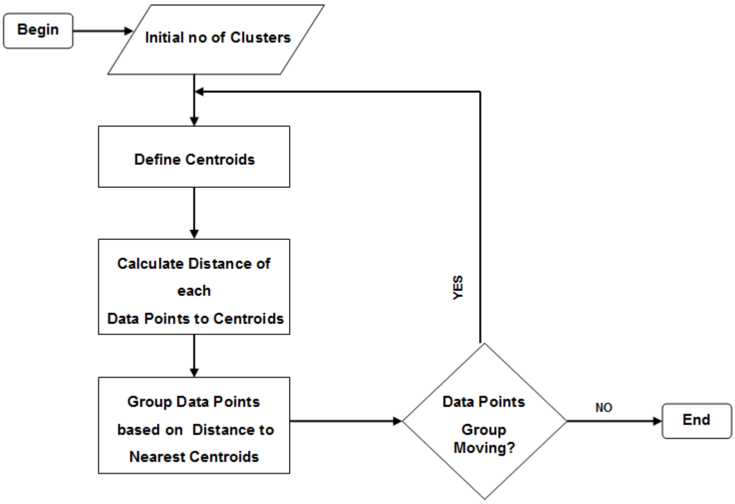

Begin
Initial no of Clusters
Define Centroids
Calculate Distance of each Data Points to Centroids
Group Data Points based on Distance to Nearest Centroids
Data Points Group Moving?
YES
NO
End

Figure 3.2: Flow chart representation of K-Means algorithm

distance defined as given below.

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ are two input vectors with n quantitative features.

3.2.3 Evaluation

In order evaluate the performance of the botnet detection techniques we need to introduce a goodness metric for a quantitative measurement. In our detection technique we basically classify the network traffic data in to normal or anomalous/suspicious groups. Any deviation from the normal traffic pattern is considered as suspicious. Hence we need to define true positive (TP), true negative (TN), false positive (FP) and false negative (FN) to determine true positive rate (TPR) and false positive rate (FPR). The below table defines TP, FP, TN and FN.

	Actual Group	Predicted Group
True Positive (TP)	Anomalous	Anomalous
False Positive (FP)	Normal	Anomalous
True Negative (TN)	Normal	Normal
False Negative (FN)	Anomalous	Normal

Now the true positive rate (TPR) which also known as senstivity and the false positive rate (FPR) can be calclulated using the following equations.

$$True\,Positive\,Rate = \frac{TP}{TP + FN}$$

$$False\,Positive\,Rate = \frac{FP}{TN + FP}$$

The true positive rate (TPR) evaluate the performance of botnet detection technique in terms of the probability of a suspicious data reported correctly as anomalous. In other words it evaluates how well the model detected anomalous packets. On the other hand the false positive rate (FPR) evaluates the performance of botnet detection technique in terms of the probability of a normal traffic reported as suspicious generating false alarms.

Summary

This chapter discussed the significance of correlation of anomalies in detecting botnets. Three different types of correlation, horizontal, vertical and cause-effect, along with the tools making use of these correlations were explained in brief. The remainder of the chapters focused on key aspects of botnet data description, namely feature selection and classification. We are making use of clustering for classifying communication pattern, which was explained with the help of k-means clustering algorithm. Finally we concluded with an evaluation technique that could be used for botnet detection techniques.

Chapter 4

Experimentation Environment

This chapter provides the details of the test setup to conduct experiments for analyzing host and network behaviors in order to detect botnets. The first section focuses on the experimental environment describing how the system and network is setup for the testing. The various tools used in order to conduct the experimentation both at the system level and network level are explained in the subsequent section.

4.1 Experimentation Setup

Any research experiments involving botnets should ensure that the bot does not unintentionally infect the computer outside the experimention setup. Failure to do so have huge implications on the security and privacy of the concerned persons and organization. Hence virtual machines are preferred to real world machines so that we would be conduct the experiment in a controlled and secure manner. In this experimentation setup we have made use of VirtualBox[37] to create virtual machines simulating real world hosts and a virtual network simulating a real world network. The details of system and network setup is provided in the following sections.

4.1.1 System Setup

The system was configured and utilities were installed in the Linux environment. As for the Linux distribution, we installed FreeBSD which is based on BSD version of UNIX. The choice of operating system was based on the excellent features such as networking, performance, security and compatibility when compared to other operating systems. Virtual machine hosts were created using VirtualBox as mentioned earlier. The virtual hosts consists of a graphical user interface (GUI) machine and command line interface (CLI) machines. The complete system setup is given below.

- Processor: Quad Core @ 3.1 GHz
- Memory: 8192 MB
- Operating System: FreeBSD 9.0-Release
- Hard disk: 350 GB
- Virtual Machine environment: VirtualBox GUI version 4.0.14_OSE
- Virtual Machine (GUI):
  - Operating System: Lubuntu 12.04
  - Memory: 1024 MB
  - Hard disk: 48 GB
- Virtual Machines (CLI):
  - Operating System: Ubuntu server 11.10
  - Memory: 128 MB
  - Hard disk: 16 GB

4.1.2 Network Setup

The network setup for the experimentation is as shown in Figure 4.1. The entire network setup is simulated using VirtualBox running on the top of the host operating system. The virtual environment consists of a number of virtual hosts as shown in Figure 4.1. One of the virtual hosts has two interfaces, one to the private network and the other to the internet via the host operating system. This virtual host machine acts as a firewall between the internal network and the internet. In order to monitor and analyze network level activities one of the virtual host machines is provided with a graphical user interface through a lightweight Ubuntu operating system called Lubuntu. The IP address of the virtual host machines are assigned as shown in the figure. If required the firewall machine can be configured to disconnect the virtual environment from the internet when analyzing the botnet.

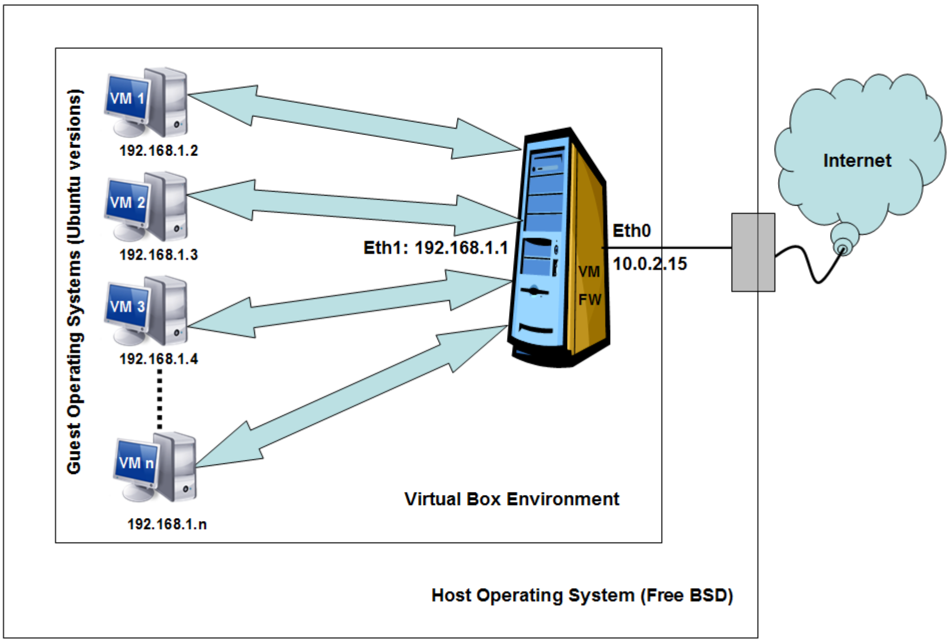


Figure 4.1: Network setup

4.2 Analysis Tools

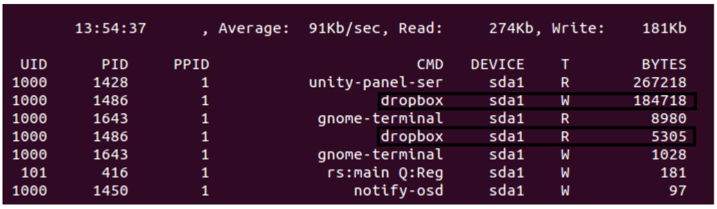
A number of tools and programs were made use in order to analyze the host behavior as well as the network behavior. These tools and programs plays an important role to accomplish the task of detecting botnets. Hence it is important to give a brief introduction to these tools and programs, discussing there advantages and limitations.

4.2.1 Strace

Strace is a useful diagnostic and debugging utility to monitor system calls and traces used by a program on Linux systems[4]. *Strace* can provide invaluable debugging information for problem solving. Information about system calls and signals happening at the kernel and user level can help us in detecting suspicious host level activities.In the context of botnet detection we could make use of *strace* to detect the programs which are executing read and write commands on the system. Strace also provides an option to monitor a running process using the *-p* option. One of the limitation of the strace tool is that at a given point of time it can monitor only a limited number of processes which is a challenge for continuous host monitoring. Hence for continuous host monitoring we make use of another tool called SystemTap[5].

4.2.2 SystemTap

SystemTap, commonly known as stap, is a tracing and probing tool that allows users to study and monitor the live activities of the computer system, especially the kernel level activities,in fine detail[5]. The tool stap provides a simple command line interface and a scripting language in which the user defines probes, actions, and data acquisition to monitor dynamically running programs. In the Linux community stap is a preferred choice for complex tasks which may require live analysis, programmable on-line response, and whole-system symbolic access[5].



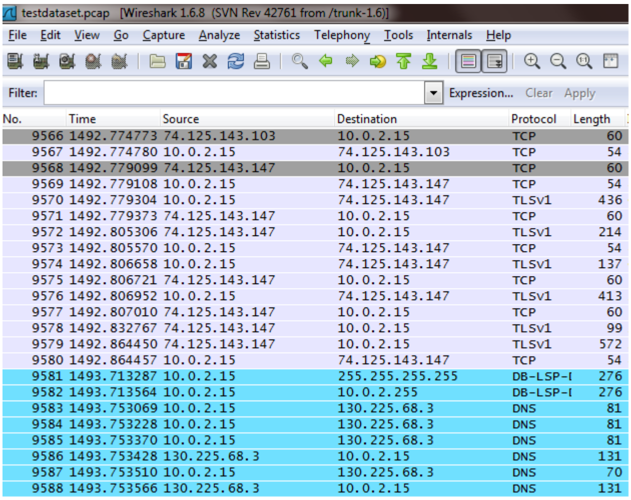
```
        13:54:37      , Average:  91Kb/sec, Read:     274Kb, Write:    181Kb

 UID      PID     PPID                 CMD   DEVICE    T        BYTES
1000     1428        1     unity-panel-ser     sda1    R       267218
1000     1486        1             dropbox     sda1    W       184718
1000     1643        1      gnome-terminal     sda1    R         8980
1000     1486        1             dropbox     sda1    R         5305
1000     1643        1      gnome-terminal     sda1    W         1028
 101      416        1       rs:main Q:Reg     sda1    W          181
1000     1450        1          notify-osd     sda1    W           97
```

Figure 4.2: SystemTap output summarizing disk read/write traffic

SystemTap offers a flexible and extendable framework by allowing users to gather important information by simply running user-written SystemTap scripts. The stap tool was designed for users with intermediate to advanced knowledge of the Linux kernel. This was one of the difficulties we faces in running this tool efficiently. We made use some of the inbuilt scripts provided by the stap team to understand the processes. An example script monitoring the disk I/O read and write commands was executed successfully. The scripts obtains the status of reading/writing disk every 5 seconds, and outputs the top ten entries during that period. Figure 4.2 shows this output where we have highlighted the dropbox processes performing the read and write activities on the disk.

4.2.3 Wireshark

Wireshark[28] is a well known network packet analyzer. The basic working principle of a network packet analyzer is to capture network packets and to display the detailed informations about the packet as much as possible. In this aspect Wireshark is well-designed and easy to use. It has a graphical user interface which enables easier interpretation of analysis results. In addition to that Wireshark has a very powerful filtering capability that understands even application level protocols. An example of Wireshark network traffic capture on our network is shown in Figure 4.3. *TShark* is a command line tool that comes along with the Wireshark tool for performing network protocol analysis. We can make use of the *tshark* to dissect the already fetched network traffic data by making use of the wide range of filtering options provided by the tool.

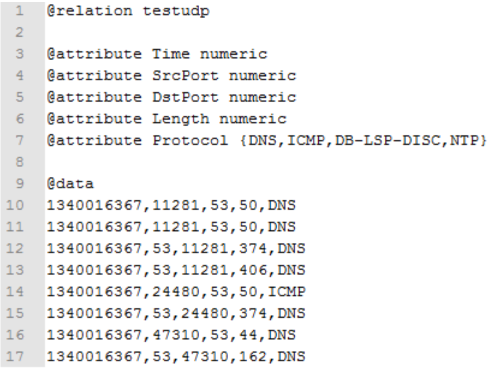

testdataset.pcap [Wireshark 1.6.8 (SVN Rev 42761 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length
9566	1492.774773	74.125.143.103	10.0.2.15	TCP	60
9567	1492.774780	10.0.2.15	74.125.143.103	TCP	54
9568	1492.779099	74.125.143.147	10.0.2.15	TCP	60
9569	1492.779108	10.0.2.15	74.125.143.147	TCP	54
9570	1492.779304	10.0.2.15	74.125.143.147	TLSv1	436
9571	1492.779373	74.125.143.147	10.0.2.15	TCP	60
9572	1492.805306	74.125.143.147	10.0.2.15	TLSv1	214
9573	1492.805570	10.0.2.15	74.125.143.147	TCP	54
9574	1492.806658	10.0.2.15	74.125.143.147	TLSv1	137
9575	1492.806721	74.125.143.147	10.0.2.15	TCP	60
9576	1492.806952	10.0.2.15	74.125.143.147	TLSv1	413
9577	1492.807010	74.125.143.147	10.0.2.15	TCP	60
9578	1492.832767	74.125.143.147	10.0.2.15	TLSv1	99
9579	1492.864450	74.125.143.147	10.0.2.15	TLSv1	572
9580	1492.864457	10.0.2.15	74.125.143.147	TCP	54
9581	1493.713287	10.0.2.15	255.255.255.255	DB-LSP-I	276
9582	1493.713564	10.0.2.15	10.0.2.255	DB-LSP-I	276
9583	1493.753069	10.0.2.15	130.225.68.3	DNS	81
9584	1493.753228	10.0.2.15	130.225.68.3	DNS	81
9585	1493.753370	10.0.2.15	130.225.68.3	DNS	81
9586	1493.753428	130.225.68.3	10.0.2.15	DNS	131
9587	1493.753510	10.0.2.15	130.225.68.3	DNS	70
9588	1493.753566	130.225.68.3	10.0.2.15	DNS	131

Figure 4.3: Wireshark example

4.2.4 Weka

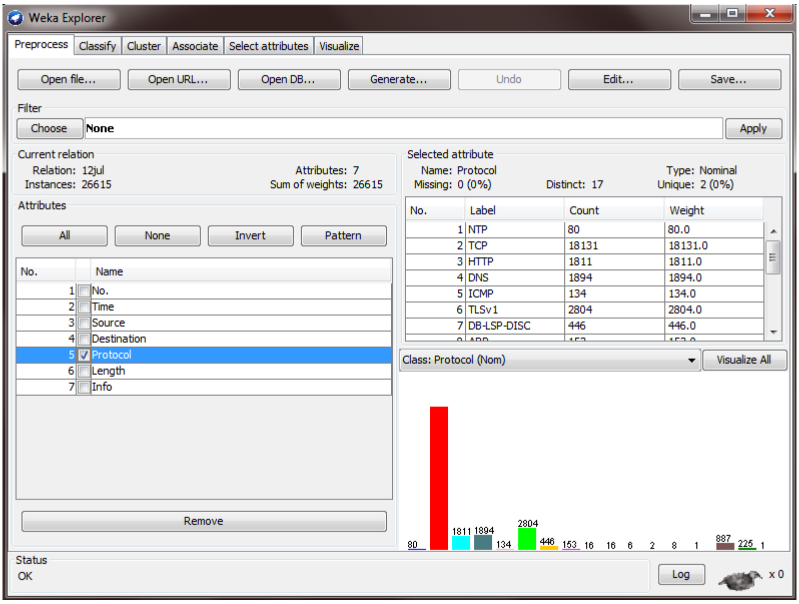
The Waikato Environment for Knowledge Analysis (Weka) is a well known open source machine learning toolkit for data mining tasks[15]. Before describing the Weka tool is important to present the file format which is used in the pre-processing and classification steps. We are making use of the Attribute-Relation File Format (ARFF), an ASCII text file, that describes a list of instances that share a set of features. ARFF files contains a header defining the attributes. This means that the internal data structures can be set up correctly before reading the actual data. Weka recommends ARFF file format. An example of ARFF file format is depicted in Figure 4.4. An ARFF file is divided into two parts, namely @relation and @data as shown in the figure. The @relation part contains a listing of attribute statements and the @data part contains all the instances of declared attributes. According to[15] the different attribute formats supported are numeric, nominal (discrete or a set of predefined values), string and date.



```
@relation testudp

@attribute Time numeric
@attribute SrcPort numeric
@attribute DstPort numeric
@attribute Length numeric
@attribute Protocol {DNS,ICMP,DB-LSP-DISC,NTP}

@data
1340016367,11281,53,50,DNS
1340016367,11281,53,50,DNS
1340016367,53,11281,374,DNS
1340016367,53,11281,406,DNS
1340016367,24480,53,50,ICMP
1340016367,53,24480,374,DNS
1340016367,47310,53,44,DNS
1340016367,53,47310,162,DNS
```

Figure 4.4: ARFF example

Weka includes algorithms for classification, clustering and various per-processing techniques enabling us to try out different machine learning algorithms on our test dataset. In addition to that is one could visualize the output results in graphical form ideal for analysis purposes. The main disadvantage of this tool is that most of the functionality is only applicable if the data is held in the main memory[15]. Another drawback being the amount of time required for processing the data. This may provide a bottleneck when analyzing large amount of data especially in real time. Since wee are mainly using Weka for clustering the the network traffic in order to train the system, these bottlenecks are not much of a problem. Weka consists of a number of different user interfaces for processing the data used to train the system. An example of Weka *Explorer* containing our test data is shown in Figure 4.5.



Filter: None

Current relation
Relation: 12jul
Instances: 26615
Attributes: 7
Sum of weights: 26615

Attributes

No.		Name
1	□	No.
2	□	Time
3	□	Source
4	□	Destination
5	☑	Protocol
6	□	Length
7	□	Info

Selected attribute
Name: Protocol
Missing: 0 (0%)
Distinct: 17
Type: Nominal
Unique: 2 (0%)

No.	Label	Count	Weight
1	NTP	80	80.0
2	TCP	18131	18131.0
3	HTTP	1811	1811.0
4	DNS	1894	1894.0
5	ICMP	134	134.0
6	TLSv1	2804	2804.0
7	DB-LSP-DISC	446	446.0

Class: Protocol (Nom)

Status
OK

Figure 4.5: Weka Explorer example

Summary

In this chapter we explained in detail about our experimentation environmet. The virtual network setup as well as the host setup were described with specifications. The subsequent section discussed about the various analysis tools we made use for our experimentation along with their use case.

Chapter 5

A Simple Prototype

The previous chapter discussed experimentation setup and various tools used to analyze the network and the host behavior. The goal of this chapter is to design a simple prototype that could detect the botnets by making use of the correlation technique. As the first step of developing such a prototype we need to train the system so that it could differentiate between benign and suspicious patterns. The training process of our system is explained in the next section. The subsequent section describes the high level system design which makes use of correlation between host and network activities to detect anomalous communication patterns.

5.1 Training the System

Training the system is referred to as the learning phase and is performed off-line by analyzing network traffic data. The basic steps involved in training the system is depicted in Figure 5.1. The accuracy of any botnet detection technique depends on how well the system is trained. The first step of the training process is to collect traffic data packets from the network. Typically this is accomplished with the help of a network monitor tool, for e.g. wireshark. Once enough network traffic data is gathered pre-processing needs to be one. This involves filtering out unwanted packets and network traffic flow analysis.

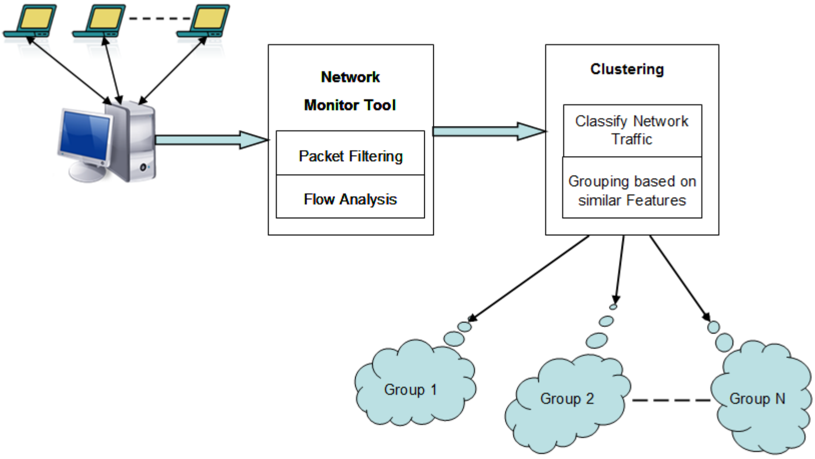


Figure 5.1: System training flowchart

The network analyzer filters out uninteresting packets or benign packets that the user trusts. Good packet filtering can highly reduce the amount of network traffic to be analyzed thereby saving a lot of time in analysis. The next step of network analyzer is to conduct flow analysis so that we can divide the network traffic based on the type of flows. This can be accomplished by making use of appropriate wireshark filters. A flow in the Internet may be defined as one or more packets traveling between two computer addresses using a particular protocol[27]. In addition to this a pair of ports is defined for each end of flow. This forms a five tuple of values $(host_{src}, host_{dest}, port_{src}, port_{dest}, protocol)$: the source and destination IP addresses, the source and destination port numbers, and the protocol identifier number, which is present in every packet.

The second stage of training is to cluster the collected network traffic and group them based on similar features. In this stage one need to select appropriate features of the network traffic in order to apply clustering algorithm. Grouping is done by making use of unsupervised learning algorithm, K-Means in this case. Once training is completed we should have a number of groups containing similar network traffic communication patterns. This result could be then used for real time analysis of network traffic by examining the incoming or outgoing network packet and comparing it with the training output.

5.1.1 Dataset Collection

In this thesis we use the dataset collected from real world network traffic through our experimentation setup. The tool wireshark is used to collect network traffic and the result was stored as *pcap* files. We generated IRC traffic we the help of Xchat[42] program as well as Dropbox[16] traffic. XChat is an IRC chat program which allows one to join multiple IRC channels (chat rooms) at the same time, for conversations, file transfers etc. Dropbox is a free cloud computing service that helps us to access our docs from anywhere through internet. Traffic generated by Dropbox is interesting because of the fact that it acts as a syncing tool, meaning that it enables users to drop any file into a designated folder that is then synced with Dropbox's Internet service to any other of the user's computers and devices with the Dropbox client. This would generate activities both at network level and host level, which could be very useful for our analysis. The data was collected during a span of twelve hours for both link directions. The summary of the traffic collected is shown in the table below.

Protocol	#Packets	%Packets	#Bytes	%Bytes
HTTP/S (TCP)	3,23,669	93.88	28,19,36,552	98.69
DNS (UDP)	15,000	4.35	26,23,802	0.92
IRC (TCP)	1396	0.40	2,76,936	0.10
DB-LSP-DISC (UDP)	772	0.23	2,12,700	0.07
NTP (UDP)	532	0.15	47,880	0.02
Others	3400	0.99	5,72,618	0.20
Total	3,44,769	100	28,56,70,488	100

The table shows the traffic collected separated using the various application level protocols. DB-LSP-DISC denotes the dropbox lan discovery protocol. The corresponding transport layer protocol is denoted in the brackets. As expected the majority of the network traffic belong to web activities as shown in the table. For the time being we are only considering TCP and UDP transport layer protocol since they represent the bulk of the network traffic.

5.1.2 Selection of Network Features

Once we have collected the dataset, the next step is to extract the network features from the dataset which is required for preprocessing and classification. We have already seen how difficult it is to select the appropriate feature and its significance in classification in chapter 3. The study conducted in[27] describes a method for extracting all the feature information by making use of the packet header information alone. They discuss the 249 network features mostly extracted from the transport protocol (TCP) header information to characterize flows. The same authors in[22] applies feature reduction techniques using correlation-based filtering method to find the best subset of features for efficient traffic classification. We could also have chosen the same subset of features, but as stated earlier these features focuses on TCP header information. Since in our analysis we are including UDP traffic, some of these features are not relevant to our analysis. The experiments conducted in[35]and [18]discusses another subset of features that are useful for classification. Some of these features were used in our analysis.

Source and destination IP addresses are commonly used traffic features for anomaly detection, especially for detection DOS attacks[20]. But since we are carrying out our experiments on virtual machine traffic and also due to the presence of NAT, source and destination IP addresses may not provide useful information in our case. Hence we have decided not to include these features for our classification. The following features were finally selected that are applicable to both TCP and UDP traffic from [22, 18]and[35] for our analysis.

- Source Port: This feature is generally used for application identification. Nowadays, more and more network traffic that uses dynamically allocated port numbe making it difficult to identify the applications. But still the feature is beneficial in figuring out suspicious network behavior for e.g. host scanning.

- Destination Port: Characteristics similar to the above feature which is also used primarily for application identification. This feature could also be used to figure out targeted attacks to specific ports, that may happen for example during worm propagation.

- Protocol (TCP/UDP): This feature makes use of the protocol identifier number defined at the transport layer. This helps us to remove any other protocol from the dataset. The information could also be used to detect abnormal network behavior, for e.g. surge of UDP traffic.

- Flow Duration: This feature information could be used to monitor communication patterns. This could be useful in detecting abnormal network behavior.

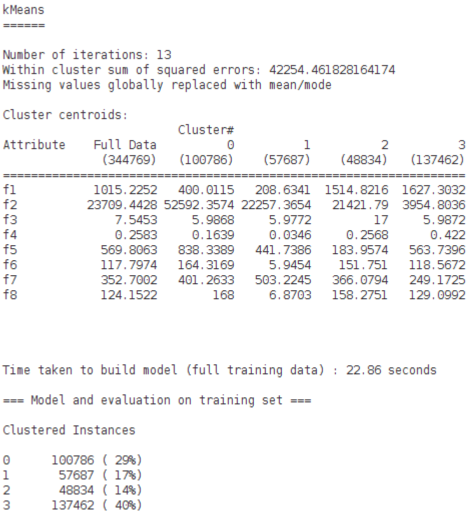
- Average packet size: This value is calculated by dividing the total size of packets in bytes received by the total number of packets received for flows. The result provides useful information about traffic statistics which can be used for classifying network behavior.

- Average packets per second: This value is calculated by dividing total number of packets received by total time elapsed between first packet and last packet for flows. The result provides useful information about traffic statistics which can be used for classifying network behavior.
- Average bytes per second: This value is calculated by dividing total size of packets in bytes received by total time elapsed between first packet and last packet for flows. This information can also be used for classifying network behavior.
- Payload length: This value can be calculated by subtracting the header length from the total packet length. This feature is important as it would give us information about the type of application running and processes carried out by those applications.
- Service: This value can extracted from the captured network traffic files for e.g., *pcap files.* This feature is important as it would give us a useful insight in interpreting the classification results.

The above mentioned features were extracted and calculated as required from the captured network traffic dataset. These results were then stored to be used by Weka tool for clustering process.

5.1.3 Clustering

Once the network features were selected and extracted from the dataset, the next step is to apply clustering algorithm on the obtained feature set. We are making use of the tool Weka[15] to accomplish the clustering procedure. Weka expects the input file either in ARFF (attribute relation file format) or CSV (comma separated values) file format. We have already explained the benefits of ARFF file format which is recommended by Weka in chapter 3. But the output of Wireshark or Tshark tools does not support ARFF format, although CSV format is supported. One of the option was to make use of the perl script provided in[27] to convert pcap files directly into ARFF files. But the method was complex and time consuming. We figured out that there was an inbuilt mechanism in Weak to convert CSV file format in to ARFF format. Hence the output generated by Wireshark/Tshark were saved in CSV file format and converted to ARFF using Weka. Once the data is ready in the appropriate ARFF format we make use of the Weka explorer to apply the required K-means clustering algorithm by selecting the initial value of the number of clusters. The experiment is repeated for different values of initial number of clusters until we get a stable result. The required result in this case is that we should be able to cluster the network traffic in to appropriate groups based on the communication patterns.



```
kMeans
======

Number of iterations: 13
Within cluster sum of squared errors: 42254.461828164174
Missing values globally replaced with mean/mode

Cluster centroids:
                         Cluster#
Attribute    Full Data          0          1          2          3
              (344769)   (100786)    (57687)    (48834)   (137462)
==================================================================
f1           1015.2252   400.0115   208.6341  1514.8216  1627.3032
f2          23709.4428 52592.3574 22257.3654   21421.79  3954.8036
f3              7.5453     5.9868     5.9772         17     5.9872
f4              0.2583     0.1639     0.0346     0.2568      0.422
f5            569.8063   838.3389   441.7386   183.9574   563.7396
f6            117.7974   164.3169     5.9454    151.751   118.5672
f7            352.7002   401.2633   503.2245   366.0794   249.1725
f8            124.1522        168     6.8703   158.2751   129.0992



Time taken to build model (full training data) : 22.86 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      100786 ( 29%)
1       57687 ( 17%)
2       48834 ( 14%)
3      137462 ( 40%)
```

Figure 5.2: Clustering example using Weka

An example output of the k-means clustering algorithm using Weka on our dataset is shown in Figure 5.2. The number of input clusters was chosen to be four. The output contains the cluster centroids of each cluster for all the selected features. The number of clustered instances of each clustered is also recorded in the output along with percentage proportions.

5.1.4 Selection of In-host Features

We need to select appropriate in-host features that would help us to detect suspicious activities in the host system. These features should be correlated with the network level activities to identify anomalous activities. Suspicious behavior on a host system can be identified by monitoring activities take place at registry, file system and network stack[43]. In our thesis we are focusing on file system activities mainly read and write operations carried out on host system. We believe this a good choice since most of the botnet code involves performing some kind of read and write operations on the infected system. These operations are carried out for downloading the malicious code on to the system, for sending confidential information to the botmaster and so on. We make use of the tool Strace[4] to inspect the file system activities on the host machine.

5.2 System Design

The basic idea behind the system design architecture is to detect anomalous activities using combined host-level and network-level information. As described in the previous chapter the experimentation setup consists of a number of host virtual machines connected to the same virtual network. Figure 5.3 shows the crux of our system design which is the co-relation engine architecture.

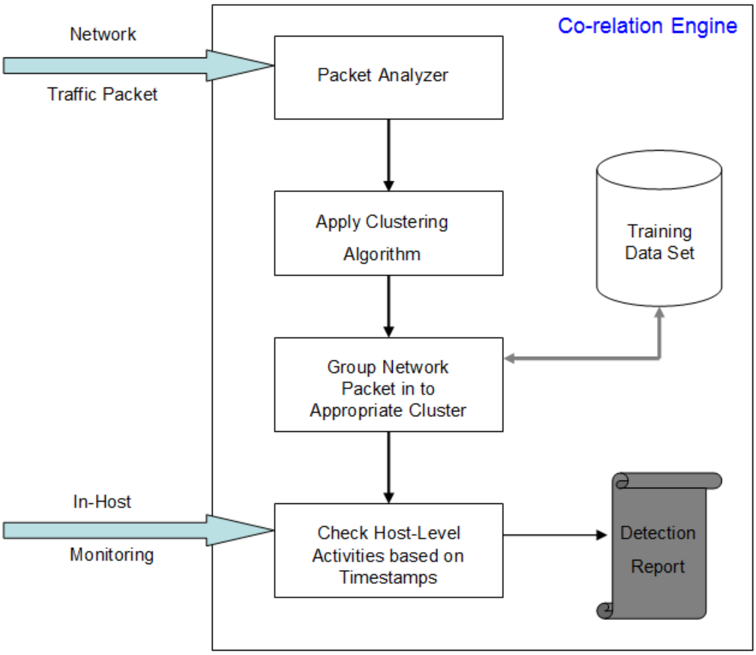

Co-relation Engine
Network
Traffic Packet
Packet Analyzer
Apply Clustering Algorithm
Training Data Set
Group Network Packet in to Appropriate Cluster
In-Host
Monitoring
Check Host-Level Activities based on Timestamps
Detection Report

Figure 5.3: Co-relation engine architecture

The simulated network consist of a monitoring host with a co-relation engine in order to analyze anomalous network activities. The architecture can be briefly explained as follows: when a new network traffic packet arrives at the monitoring host the network packet analyzer will fetch the required information that's needed by the clustering algorithm as input. The packet analyzer converts the required information in to the appropriate format for the clustering algorithm. The unsupervised K-Means clustering algorithm is applied on the network packet information to calculate the distance between the newly arrived network packet and the centroids of already grouped network traffic patterns.

This is accomplished with the help of information available from the training data set which already contains grouped traffic clusters. Based on the distance result the network packet is grouped in to the nearest cluster. After clustering the next step is to get the host level activities for correlating with network activities.

Each host in the network will have a host analyzer in the form of a in-host monitor capable of monitoring system level activities. This in-host monitor will be used in turn to detect suspicious activities. A crucial task of correlation engine in botnet detection is to compare the clustering analysis obtained from the network analyzer with the output of in-host monitor to identify similar communication activities that look suspicious. For e.g. read/write commands on the system. The comparison is done based on timestamps. This is based on our assumption that all the systems in the network have synchronized clock times. The time window needed to be selected accordingly. The idea here is that if we could correlate activities happening both at the host and in the network approximately at the same time we could conclude that they are related activities. This is highly useful in the case of botnet detection because of the fact that bots in a botnet co-ordinates similar and related activities both at host and network level.

5.2.1 Packet Analyzer

In this section we discuss the high-level design of our packet analyzer which is shown in the Figure 5.4. The input to the packet analyzer are the packets that are flowing in the network interface that is set to promiscuous mode. The packet analyzer need to store the incoming packets in the buffer for processing. Appropriate data structures are defined in order accomplish this task. The next step would be process this stored packets by striping out the required network and transport header information. The information collected will be stored in temporary variables and will be used in computing network statistics if required. The final step of the packet analyzer is to output the result in to a file which would be used by clustering program for classification.

5.2.2 Clustering

In this section we discuss the k-means clustering algorithm used for implementation. A good implementation of k-means algorithm for botnet detection is discussed in[25]. But we need to change the algorithm slightly to achieve the

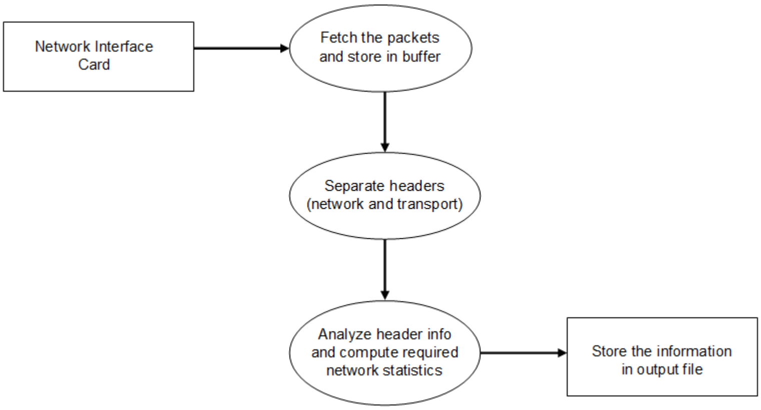
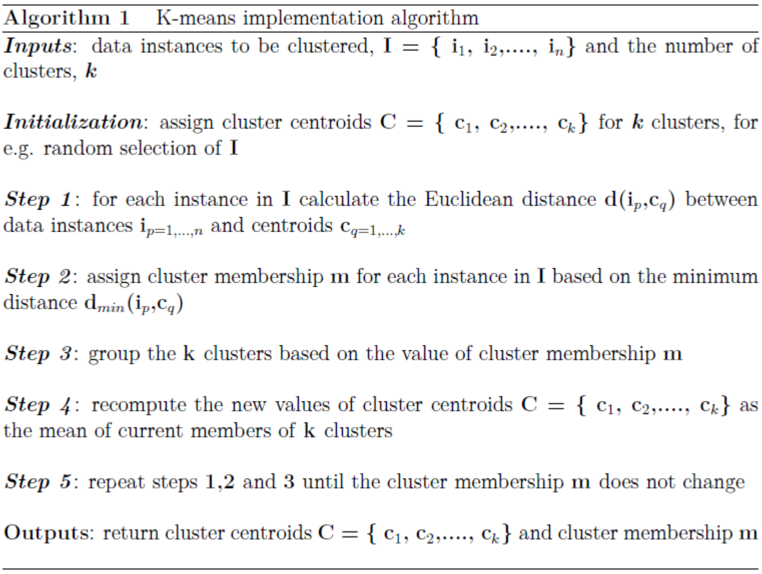

Network Interface Card
Fetch the packets and store in buffer
Separate headers (network and transport)
Analyze header info and compute required network statistics
Store the information in output file

Figure 5.4: High-level design of packet analyzer

desired result. The algorithm used for our k-means implementation is shown in Figure 5.5. The input to the algorithm is the data instances containing the network features from our packet analyzer and the no of clusters required. We initialize the values of clustering centroids with those values obtained from the training dataset. The algorithm outputs the new cluster centroids and the cluster membership of the network packet processed. The time complexity of k-means algorithm is calculated approximately linear $O(N)$, where N is the number of instances to be clustered[25]. This makes it a suitable choice for clustering large datasets, especially when the number of clusters are previously known.

5.2.3 In-host Monitoring

In this section we discuss our in-host monitoring strategy to detect suspicious host level activities. As mentioned earlier we make use of the tools SystemTap[5] and Strace[4] to accomplish this task. The monitoring is done based on the timestamps received from the network packet information. We are assuming that the clocks are synchronized through out all the systems in the network. This is a reasonable assumption in our test environment. To start the process we have to define a time window to carry out the in-host monitoring. We have decided to monitor the system for every 5 seconds and collect the required information. The tool SystemTap is used for this continuous monitoring and the information about the processes executing disk I/O read and write commands were recorded. We then make use of process id of the applications executing the



```
Algorithm 1   K-means implementation algorithm
Inputs: data instances to be clustered, I = { $i_1$, $i_2$,...., $i_n$} and the number of clusters, k

Initialization: assign cluster centroids C = { $c_1$, $c_2$,...., $c_k$} for k clusters, for e.g. random selection of I

Step 1: for each instance in I calculate the Euclidean distance $d(i_p,c_q)$ between data instances $i_{p=1,...,n}$ and centroids $c_{q=1,...,k}$

Step 2: assign cluster membership m for each instance in I based on the minimum distance $d_{min}(i_p,c_q)$

Step 3: group the k clusters based on the value of cluster membership m

Step 4: recompute the new values of cluster centroids C = { $c_1$, $c_2$,...., $c_k$} as the mean of current members of k clusters

Step 5: repeat steps 1,2 and 3 until the cluster membership m does not change

Outputs: return cluster centroids C = { $c_1$, $c_2$,...., $c_k$} and cluster membership m
```

Figure 5.5: K-means algorithm

disk I/O read and write commands to fetch more information about the various system calls carried out by this processes. This is accomplished by passing this process id to the strace tool for dissection of the process in consideration. Strace tool make use of the *-p* option to accomplish this. The strace output should be examined thoroughly to understand the various system calls happening and the significance for disk I/O read and write operation carried out by the process. A detailed examination of the strace output is provided in the next chapter.

Summary

This chapter discussed the details about our simple prototype used for the correlation analysis. We explained how the test dataset were collected with statistics. Selecting network and in-host features are the key aspects of any training system. The selected network and in-host features were explained with the reason behind their selection. The final part of the chapter contains the description of our system design in detail by explaining the important parts namely, packet analyzer, clustering and in-host monitoring.

Chapter 6

Analysis

This chapter provides a practical analysis of our simple prototype system by analyzing the dataset collected using our experimentation setup. The first section discusses how the results obtained from clustering network traffic is analyzed. In the subsequent section we discuss how we are correlating the result obtained from network traffic analysis with the read and write system call events on the host to detect suspicious activities. We conclude our analysis with a section describing the lessons we have learned from this analysis.

6.1 Analysis of Network Traffic

The network traffic analysis is carried out by inspecting the clustering output we obtained using the Weka tool. Weka explorer has a GUI to visualize clustering output in 2-dimesion as shown in Figures 6.1 and 6.2. Figure 6.1 shows the clustering output in relation with source port and Figure 6.2 shows the output relation in relation with destination port. The cluster output depicts how the various application level protocols are scattered within the clusters. The dropbox lan discovery protocol could be easily detected from the output as it used the same port number for source and destination. The IRC protocol forms a small group in both figures. But it also interesting to note that some of the IRC traffic have been scattered across cluster 2 in Figure 6.1 and across cluster 3 in Figure 6.2. Although we could get some useful information from the clustering output we were not sure how good these results are in detecting anomalous traffic patterns. Also since the results are represented in 2-dimension it is difficult to make any clear cut conclusions. If we could plot the output in 3 or more dimensions the output could be interpreted in a more meaningful manner.

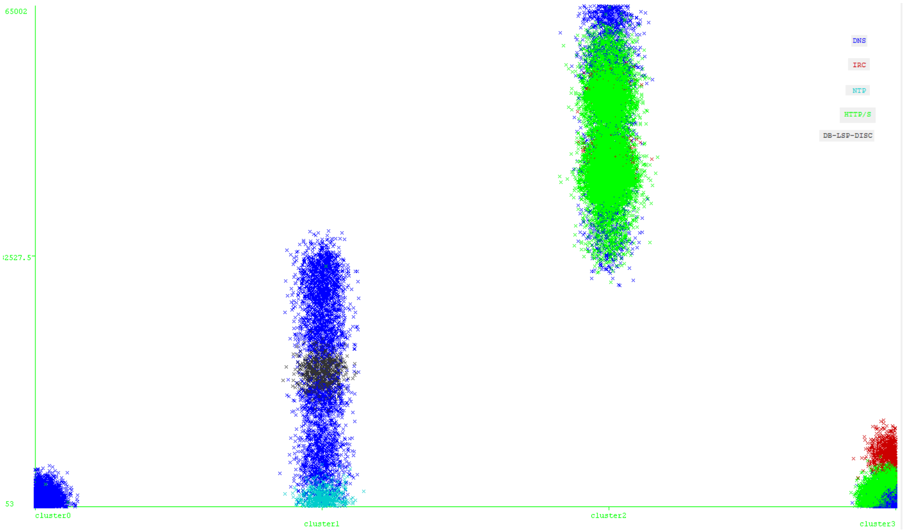


Figure 6.1: Clustering output with Y-axis: Source Port

After careful inspection of all the clustering output in 2-dimesion across all features we found that the clustering output plot of protocol against payload length provided an interesting output which is worth discussing in detail. The clustering output is shown in the Figure 6.3.

Figure 6.3 depicts the clustering output of application protocols based on the payload length.The dropbox lan discovery, NTP and DNS protocols does not vary much with respect to the payload length with our test data set. The web traffic is spread across the pay load length till mid of the graph and an small group at the top of the graph denoting increased payload. The increased payload in web traffic must be due to the files downloaded by the browser automatically or by the user. The most interesting aspect is the clustering output of of IRC traffic. Most of the IRC traffic has a small payload length as expected since they involves chat messages. Chat messages generally are short in nature and hence it make sense that most of the IRC traffic are grouped at the beginning of the y-axis. On the other hand there is a group of IRC traffic belonging to

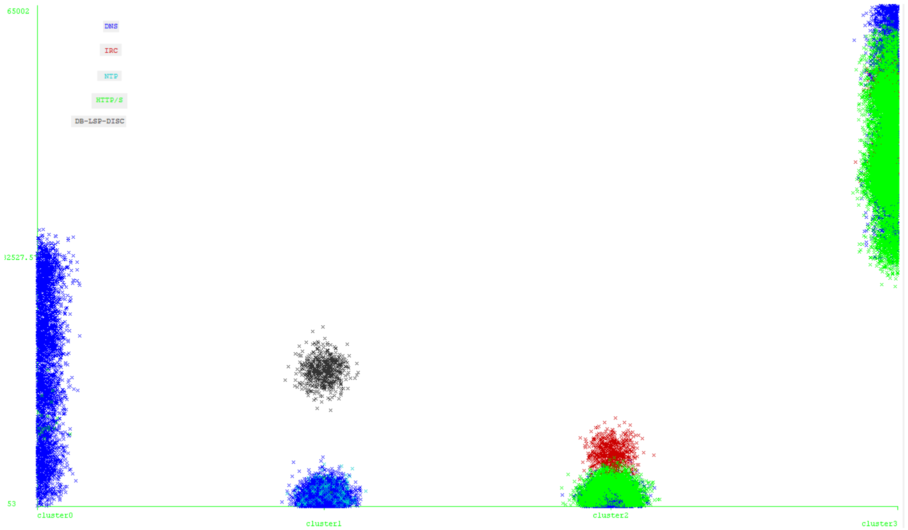


Figure 6.2: Clustering output with Y-axis: Destination Port

much higher payload that could raise our eyebrows in suspicion. One of such packet is marked using a black circle as shown in Figure 6.3.

We need to further drill down the clustered output to make any reasonable conclusion. Weka provides us an option to accomplish this by providing an user interface to fetch the information about the traffic packet in consideration. The following table shows the information fetched from Weka for the marked instance.

Instance number	305786
Source Port	6667
Destination Port	54363
Length	1474
Protocol	IRC
Time	19:08:47.519652
Cluster	cluster3

The source port and destination port denotes the port numbers used by the IRC instance. Higher payloads in IRC traffic may be an indication of some kind of file downloading or uploading happening in the host system. The cluster number information could be used to mark that cluster as suspicious for future analysis. For e.g. if the analysis of future IRC traffic with high payload also belongs to

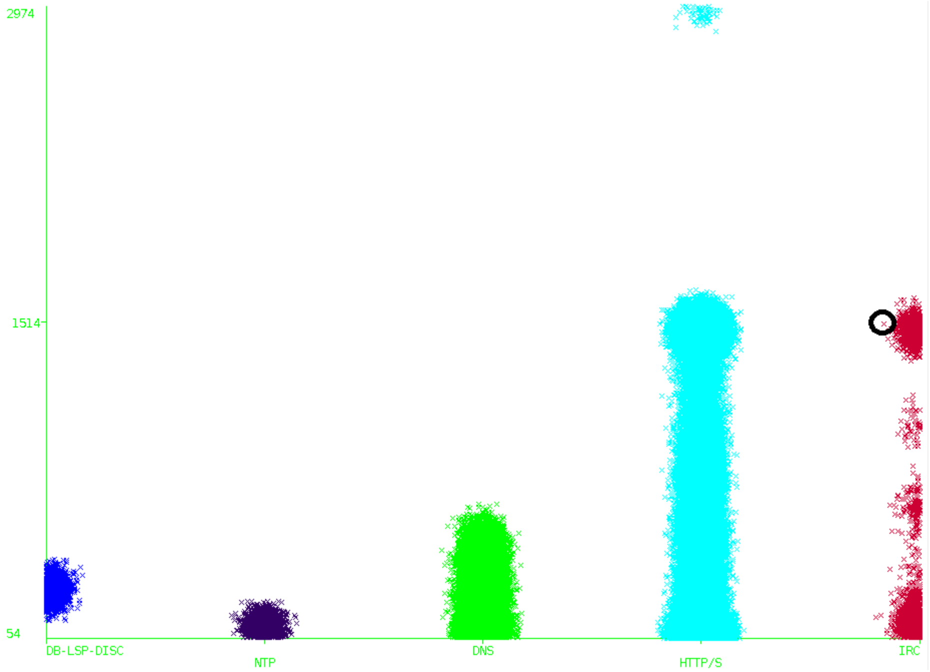
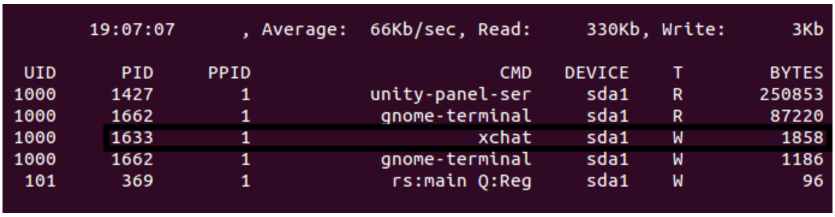


Figure 6.3: Clustering output with Y-axis: Payload Length

cluster 3, then it would be much easier for us to detect anomalous traffic. The timestamp of the packet will be used as input to the in-host monitoring system for correlating system calls. To gather further information about the suspicious instance we have to analysis the events happening at the host system.

6.2 Correlation with in-host events

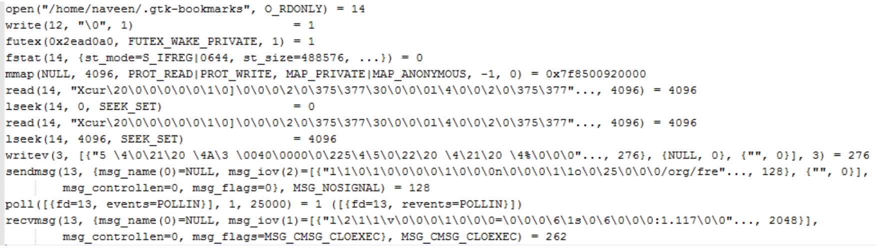
The information gathered from the network traffic analysis is in turn used to analyze the host systems. The correlation is done based on timestamps under the assumption that clocks are synchronized in the network. The timestamp of the required instance for inspection is passed to the in-host monitor from the output of network traffic analyzer. In-host monitoring of processes is carried out by SystemTap tool. The tools make use of a script that obtains the status of processes reading/writing disk every 5 seconds. The output provides us useful information such as the process id (*pid*), the name of the processes that carried out disk read and write operation during that time period and the amount of bytes read and written by the processes. Figure 6.4 shows the Systemtap output for the provided timestamp.



```
      19:07:07      , Average:  66Kb/sec, Read:     330Kb, Write:      3Kb

 UID       PID      PPID                  CMD   DEVICE     T        BYTES
1000      1427         1      unity-panel-ser     sda1     R       250853
1000      1662         1       gnome-terminal     sda1     R        87220
1000      1633         1                xchat     sda1     W         1858
1000      1662         1       gnome-terminal     sda1     W         1186
 101       369         1         rs:main Q:Reg     sda1     W           96
```

Figure 6.4: SystemTap output

From the Figure 6.4 it is clear that the program responsible for IRC traffic is *xchat* and it perform a write operation of disk consisting of 1858 bytes. The next step of our analysis is to trace the *xchat* program to identify the various system calls executed by it. The SystemTap output provides us with the process id (PID) of the *xchat* program. This information is given to the strace tool to trace the process. The command executed is *"strace -p 1633 -o stracexchat.out"*. Figure 6.5 shows the snapshot of the output file with some details. The *write( )* and *writev( )* function attempts to write the information from the buffer to the file associated in the command. This first line performs the system call *open( )* on the file "/home/naveen/.gtk-bookmarks". The file shall be opened read-only (*O_RDONLY*). This call returns the file descriptor 14 to the file opened. The *fstat( )* command in the output tells that the file number 14 is a regular file. The general usage of *mmap( )* is to allocate more memory for the process. The malloc and calloc library functions usually use it internally. Having looked at output we could conclude that various file operations are carried out and messages are communicated in both direction. This would be a good reason for being suspicious but not enough to say that it is malicious.



```
open("/home/naveen/.gtk-bookmarks", O_RDONLY) = 14
write(12, "\0", 1)                          = 1
futex(0x2ead0a0, FUTEX_WAKE_PRIVATE, 1) = 1
fstat(14, {st_mode=S_IFREG|0644, st_size=488576, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8500920000
read(14, "Xcur\20\0\0\0\0\0\1\0]\0\0\0\2\0\375\377\30\0\0\01\4\0\0\2\0\375\377"..., 4096) = 4096
lseek(14, 0, SEEK_SET)                      = 0
read(14, "Xcur\20\0\0\0\0\0\1\0]\0\0\0\2\0\375\377\30\0\0\01\4\0\0\2\0\375\377"..., 4096) = 4096
lseek(14, 4096, SEEK_SET)                   = 4096
writev(3, [{"5 \4\0\21\20 \4A\3 \0040\0000\0\225\4\5\0\22\20 \4\21\20 \4%\0\0\0"..., 276}, {NULL, 0}, {"", 0}], 3) = 276
sendmsg(13, {msg_name(0)=NULL, msg_iov(2)=[{"l\1\0\1\0\0\0\0\1\0\0\0n\0\0\0\1\1o\0\25\0\0\0/org/fre"..., 128}, {"", 0}],
        msg_controllen=0, msg_flags=0}, MSG_NOSIGNAL) = 128
poll([{fd=13, events=POLLIN}], 1, 25000) = 1 ([{fd=13, revents=POLLIN}])
recvmsg(13, {msg_name(0)=NULL, msg_iov(1)=[{"l\2\1\1\v\0\0\0\1\0\0\0=\0\0\0\6\1s\0\6\0\0\0:1.117\0\0"..., 2048}],
        msg_controllen=0, msg_flags=MSG_CMSG_CLOEXEC}, MSG_CMSG_CLOEXEC) = 262
```

Figure 6.5: Snapshot of strace output

6.3 Lessons Learned

Based on our experience detecting anomalous traffic by combining network-level and host-level information is challenging as well as interesting. In this section we describes the various challenges we faced while performing the analysis and the lessons learned from these challenges. The most difficult as well as time consuming task was to finalize the network features to be extracted to perform the system training. As mentioned earlier in[27] the authors extracts 249 network features of which majority are extracted when TCP header information. Since we are taking UDP traffic also in to consideration the total number of features would be even more. Filtering this network features to figure out the best possible candidates is an herculean task. Moreover experiments needed to be performed by selection various combination of these features to come up with a precise feature task. Many researches have been conducted on the same and it seems not to be trivial task. We decided to take a combination of features from some of these research papers that seems to have better results than others. We are not aware of any foolproof method to select the best features to perform botnet detection. Extracting relevant features from pcap files also was not an easy task. We made use of the fullstats tool mentioned in[22] to extract some of the key features.

Once the features are selected the next hurdle is to train the system using a good neural network algorithm until we get a stable result. We decided to use clustering for classifying network traffic using the tool Weka. The simple K-means algorithm implementation provided by Weka was chosen to train the system. The difficult task with K-means algorithm is to determine the optimum no of clusters required for training the system. Weka has a nice user interface which helps you to visualize the clustering outputs. But on the other side we figured out that the tool was a heavyweight process in terms of memory consumption. The Weka explorer loads the data instance in to main memory in order to perform clustering. Hence when the no of data instances increased the system performance was effected badly even resulting in system crash or freezing. Since we were running these experiments in the virtual machine did not help the matter either. In order to overcome this problem we had to allocate more memory for the virtual machine during the training phase. This is not an ideal solution and can handle only data instance less than one million. We gathered some valuable information during initial training phase but the results were not consistent enough to draw any valuable conclusion. We believe a better normalization technique on the selected features would have produced a much better result.

Next let us discuss some of the problems we faced in setting up in-host monitoring for analysis. Initially we decided to make use of the *strace* process to perform the in-host monitoring. But later we found that it is a complicated task to perform continuous monitoring using *strace*. Also there is a limit on the number of process one could attach to *strace* tool. Even more challenging was to figure out which processes are required to be traced for further inspection. Hence we decided to use the SystemTap tool to perform continuous monitoring. But installation of SystemTap was not straightforward as we thought. Also the tool was designed for users with intermediate to advanced knowledge of the Linux kernel. We managed install the SystemTap tool with the help of online manual and user groups. Installation required kerned debug packages in order to run system tap scripts correctly. Writing system tap scripts was also not an easy task and hence we made use of readily available online scripts. Once we figured out the processes to be traced from SystemTap output, we made use of *strace* for in depth inspection.

Summary

In this chapter we discussed our analysis results gathered from the experiments conducted. To begin with we provided a brief discussion on the analysis of network traffic calls with the help of clustering outputs. The information collected from the Weka tool was also provided. In the following section we stated how correlation was performed by tracking system calls based on timestamps. The final part of chapter discussed the various challenges we faced during this analysis and the lessons we learned.

Chapter 7

Conclusion

Botnets are considered as the biggest threat to internet security today. Day by day millions of computers are compromised on the internet. These compromised computers are controlled by botmasters to launch all kinds of malicious activities resulting in sophisticated attacks across the web. Moreover botnet controllers are becoming more and more clever by incorporating latest evasion techniques thereby making it even harder to detect and defend against botnets. Hence we need to come up with better detection solutions to mitigate botnets.

A large number of methods have been proposed in the literature for detection and elimination of botnets. The focus of this thesis project was to exploit a method which detects botnets by combing information gathered from network-level and host-level activities. The basic idea behind this approach was to identify patterns of anomalous behavior which develop in similar ways across the network, such as within a sub-set of computers within a given sub-net. The identified patterns were then correlated with system level activities for gather further information required to detect botnets.

A suitable platform and operating system was selected and setup in order to perform the analysis. The dataset required for our analysis was collected in real time using our virtual environment setup. The next task of our thesis project was to select appropriate network features in order to train the system. Selecting good network features to differentiate network traffic is a challenging task.

After going through a few research papers we decided to chose a combination of network features selected from these research papers.

We trained our system using the selected network features by selecting simple k-means clustering algorithm in the data mining tool Weka. Although we got some valuable results during initial training phase the results lacked stability. Hence we were not able to formulate any meaningful conclusion. We are not sure whether this was the problem with the selected features or a problem with the quality of raw data collected from the test environment. However we do believe that a better normalization technique could have produced much better analysis results. Also a better and efficient implementation of k-means algorithm for clustering process would have given us a higher quality output. For example we made use of Euclidean distance which resulted in spherical clusters. It is worth to have a look in to other distance measures, for e.g. Manhattan distance. The clustering output were visualized in 2-dimension using Weka's explorer interface. We realized that it is hard to interpret the results in two dimension since we have selected more than two features as input to the clustering algorithm.

In-host monitoring was carried out by monitoring processes executing disk I/O read and write commands. According to us this was a reasonable choice since most of the botnets performed some kind of read/write system calls on the infected host. A better result could have been achieved if we could monitor kernel level system and registry calls. Correlation was done using timestamps based on the assumption that the clocks are synchronized on all hosts. Although this was possible with our test setup we think that it will be really a difficult task to ensure this in the real world.

Detecting botnets by correlating network-level and host-level activities is a challenging as well as interesting research area. We tried to develop a prototype by correlating network-level and host-level information we gathered from our training phase. This aim was to correlate identified network patterns with host-level disk I/O read and write operations to detect anomalous activities. We were able to obtain some valuable results but was not enough to come up with a precise conclusion. However we have learned that by fine-tuning the selected network features and applying a better version of k-means algorithm would have resulted in a more meaningful detection result.

7.1 Future Work

During the training phase we recognized that the challenge of finding the best possible combination of features and machine learning algorithms remains highly complicated and deserves further investigation. Most of the features selected were application specific and requires more training on large datasets to produce meaningful results. We would like to experiment with some of the other features mentioned in[27] for further analysis and evaluation. Weka tool provides a wide range of choice for classification and clustering algorithms. In the future we plan to apply a different machine learning clustering algorithm DBSCAN[7] on the selected features and evaluate the results. We think that results would be interesting since DBSCAN algorithm relies on a density based notion of clusters.

Summary

Botnet detection is generally difficult and still is a hot topic in the research community. In this thesis we tried to develop a prototype to detect botnets by combining network-level and host-level detection techniques. Even though we did not come up with a precise answer to our research goal, we have learned a lot with respect to the botnet detection techniques, the significance of machine learning algorithms in training botnet detection systems and how we can combine these techniques to achieve a multi-perspective view thereby enabling better detection results.

Bibliography

[1] Mohammed S. Alam and Son T. Vuong. Advanced methods for botnet intrusion detection systems. In *Intrusion Detection Systems*, 2011. http://www.intechopen.com/books/intrusion-detection-systems/advanced-methods-for-botnet-intrusion-detection-systems.

[2] Adam J. Aviv and Andreas Haeberlen. Challenges in experimenting with botnet detection systems. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET'11)*, August 2011.

[3] Paul Barford and Vinod Yegneswaran. An Inside Look at Botnets. In Mihai Christodorescu, Somesh Jha, Douglas Maughan, Dawn Song, and Cliff Wang, editors, *Malware Detection*, volume 27 of *Advances in Information Security*, chapter 8, pages 171–191. Springer US, Boston, MA, 2007.

[4] O. S. Community. strace. *http://sourceforge.net/projects/strace/*, 2009.

[5] Frank C. EIgler, Vara Prasad, Will Cohen, Hien Nguyen, Martin Hunt, Jim Keniston, and Brad Chen. Architecture of systemtap: a linux trace/probe tool, 2005.

[6] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, MineNet '06, pages 281–286, New York, NY, USA, 2006. ACM.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, 1996.

[8] Jason Franklin, Mark Luk, Jonathan M. Mccune, Arvind Seshadri, and Leendert Van Doorn. Towards sound detection of virtual machines. In *In Springer Book on Botnet Research*, 2007.

[9] Guofei Gu. *Correlation-based botnet detection in enterprise networks.* PhD thesis, Atlanta, GA, USA, 2008. AAI3327579.

[10] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association.

[11] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *USENIX Security Symposium*, pages 139–154, 2008.

[12] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[13] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[14] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *NDSS*, 2008.

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[16] Dropbox Inc. Dropbox, https://www.dropbox.com/.

[17] R.A. Kemmerer and G. Vigna. Intrusion Detection: A Brief History and Overview. *IEEE Computer*, pages 27–30, April 2002. Special publication on Security and Privacy.

[18] A. Kind, M. P. Stoecklin, and X. Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Trans. on Netw. and Serv. Manag.*, 6(2):110–121, June 2009.

[19] Igor Kononenko and Matjaz Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms.* Horwood Publishing Limited, West Sussex.

[20] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, August 2005.

[21] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4):227–261, 2000.

[22] W. Li and A. W. Moore. A machine learning approach for efficient traffic classification. In *Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '07, pages 310–317, Washington, DC, USA, 2007. IEEE Computer Society.

[23] Yingqiu Liu, Wei Li, and Yun-Chun Li. Network traffic classification using k-means clustering. In *Proceedings of the Second International Multi-Symposiums on Computer and Computational Sciences*, IMSCCS '07, pages 360–365, Washington, DC, USA, 2007. IEEE Computer Society.

[24] Yingqiu Liu, Wei Li, and Yun-Chun Li. Network traffic classification using k-means clustering. In *Proceedings of the Second International Multi-Symposiums on Computer and Computational Sciences*, IMSCCS '07, pages 360–365, Washington, DC, USA, 2007. IEEE Computer Society.

[25] Wei Lu, Goaletsa Rammidi, and Ali A. Ghorbani. Clustering botnet communication traffic based on n-gram feature selection. *Comput. Commun.*, 34(3):502–514, March 2011.

[26] Wei Lu, Mahbod Tavallaee, and Ali A. Ghorbani. Automatic discovery of botnet communities on large-scale communication networks. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 1–10, New York, NY, USA, 2009. ACM.

[27] Andrew Moore, Michael Crogan, Andrew W. Moore, Queen Mary, Denis Zuev, Denis Zuev, and Michael L. Crogan. Discriminators for use in flow-based classification. Technical report, 2005.

[28] Angela Orebaugh, Gilbert Ramirez, Josh Burke, and Larry Pesce. *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[29] Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. *ACM Trans. Program. Lang. Syst.*, 30(5):25:1–25:54, September 2008.

[30] R. Puri. Bots & botnet: An overview. *SANS Institute?03*, 2003.

[31] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *LISA*, pages 229–238. USENIX, 1999.

[32] Craig Schiller and Jim Binkley. *Botnets: The Killer Web Applications.* Syngress Publishing, 2007.

[33] Seungwon Shin, Zhaoyan Xu, and Guofei Gu. Effort: Efficient and effective bot malware detection. In *INFOCOM*, pages 2846–2850, 2012.

[34] L. Spitzner. *Honeypots: Tracking Hackers.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[35] W. Timothy Strayer, David E. Lapsley, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In *Botnet Detection*, pages 1–24. 2008.

[36] Amit Kumar Tyagi and G.Aghila. Article: A wide scale survey on botnet. *International Journal of Computer Applications*, 34(9):10–23, November 2011. Published by Foundation of Computer Science, New York, USA.

[37] Jon Watson. Virtualbox: bits and bytes masquerading as machines. *Linux J.*, 2008(166), February 2008.

[38] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 232–249, Berlin, Heidelberg, 2009. Springer-Verlag.

[39] Xiaonan Zang, Athichart Tangpong, George Kesidis, and David J Miller. Botnet detection through fine flow classification. *Science*, (0915552):1–17, 2011.

[40] Hossein Rouhani Zeidanloo and Azizah Bt Abdul Manaf. Botnet detection by monitoring similar communication patterns. *CoRR*, abs/1004.1232, 2010.

[41] Hossein Rouhani Zeidanloo, Mohammad Jorjor Zadeh, and Mazdak Zamani M. Safari. A taxonomy of botnet detection techniques.

[42] Peter Zelezny. Xchat, http://www.xchat.org/.

[43] Yuanyuan Zeng, Xin Hu, and Kang G. Shin. Detection of botnets using combined host- and network-level information. In *DSN*, pages 291–300, 2010.

[44] Zhaosheng Zhu, Guohan Lu, Yan Chen, Zhi J. Fu, Phil Roberts, and Keesook Han. Botnet Research Survey. In *Computer Software and Applications, 2008. COMPSAC '08*, pages 967–972, July 2008.