5-2009

# Covert Botnet Implementation and Defense Against Covert Botnets

Lokesh Babu Ramesh Babu
*Utah State University*

# COVERT BOTNET IMPLEMENTATION AND DEFENSE AGAINST COVERT BOTNETS

by

Lokesh Babu Ramesh Babu

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

| | |
|---|---|
| _____ | _____ |
| Chad Mano<br>Major Professor | Donald H. Cooley<br>Committee Member |
| _____ | _____ |
| Xiaojun Qi<br>Committee Member | Byron R. Burnham<br>Dean of Graduate Studies |

UTAH STATE UNIVERSITY
Logan, Utah

2009

# ABSTRACT

Covert Botnet Implementation and Defense Against Covert Botnets

by

Lokesh Babu Ramesh Babu, Master of Science

Utah State University, 2009

Major Professor: Dr. Chad Mano
Department: Computer Science

The advent of the Internet and its benevolent use has benefited mankind in private and business use alike. However, like any other technology, the Internet is often used for malevolent purposes. One such malevolent purpose is to attack computers using botnets. Botnets are stealthy, and the victims are typically unaware of the malicious activities and the resultant havoc they can cause. Computer security experts seek to combat the botnet menace. However, attackers come up with new botnet designs that exploit the weaknesses in existing defense mechanisms and, thus, continue to evade detection.

Therefore, it is necessary to analyze the weaknesses of existing defense mechanisms to find the lacunae in them and design new models of bot infection before the attackers do so. It is also necessary to validate the analysis and the design of such a model by implementing the attack and fine-tuning the design. This thesis validates the weaknesses found in existing defense mechanisms against botnets by implementing a new model of botnet and carrying out experiments on it.

To merely analyze and present the weaknesses of a defense would open the door for attackers and make their job easier. Thus, creating a defense mechanism against the new attack is equally important. This thesis proposes a design against the new model of bot infection and also implements the design. Experiments were conducted to validate and fine-tune the design and eliminate flaws in the new defense mechanism.

(89 pages)

# ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Chad Mano, for his invaluable guidance through the course of my research. The computer security course I took from him kindled my interest and eventually led me to pursue research in this field. His ideas, experience, and friendly attitude have immensely benefited my research. I would also like to express my gratitude to Dr. Don Cooley for his constant support and encouragement as I have pursued my research. My heartfelt thanks go to Dr. Xiaojun Qi for her feedback and ideas. Huge thanks to Myra Cook whose feedback helped to make my thesis presentable.

I will be eternally grateful to my family, Ramesh Babu, Manjula Ramesh, Venkatesh, Krishna Murthy, and Sathish Babu, for their selfless love and support.

I would like to thank Brandon Shirley for helping me understand the design he proposed in his research.

Lokesh Babu Ramesh Babu

CONTENTS

Page

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

The Internet is a global system of computer networks that enable transfer of information in the form of web pages, email, and other messaging services between users in different parts of the world. Prior to 1990, it was the exclusive domain of tech savvy people, such as educators, and researchers; however, now it is being used both by tech savvy people as well as people with less technological expertise. Companies have set up websites on the Internet to provide services to their customers much more effectively. People from all walks of life commonly use the Internet for such activities as buying products, communicating with friends, and performing online banking within the comfort of their homes. Not only are billions of dollars transacted everyday online, but people also give out personal information online when they use Internet.

E-commerce websites make huge money through online trading. When the Internet was restricted to only educational and research institutions, computer security was not a huge issue; however, with the current widespread use of the Internet, computer security has become very important. Easy access to the Internet has benefited people in a large way, but unfortunately, it has also given wider scope for attackers to target computers and cause huge damage. If businesses' websites providing services for their customers were to go down, the business entities would suffer huge financial losses as a result of the denial-of-service to users.

The Internet has also led to new types of advertising models, such as pay-per-click. In this type of model, websites display advertisements in their web pages. When a

user clicks on these advertisements, the advertiser pays the websites hosting these advertisements. Unfortunately, this innovation has led to a new model of fraud, known as the click fraud. In this type of fraud, an automated program clicks on the ad to generate a charge per click. The website displaying these ads can easily create the automated program to get greater revenue.

Communication has been revolutionized with the advent of Internet; there has been a huge increase in the use of email for both private and professional purposes alike over the years. Again, the widespread use of email has led to devious ways of exploiting such usage. Unsolicited email, commonly referred to as spam, is also on the rise. Spam overloads email inboxes with worthless messages, causing annoyance. Even more damaging, Spam also harms Internet service providers by overloading mail servers, leading to nondelivery of legitimate email. Additionally, the Internet service provider may get blacklisted because of spammers who misuse their service to send spam. This leads to a loss of business as well as the loss of the reputation of an Internet service provider.

A botnet [1, 2, 3] is one of the techniques used for execution of malicious activities including denial-of-service, click fraud, identity theft, and spamming attacks. A botnet is a collection of compromised computers in a network. The compromised computer has a malicious application known as a bot. There can be tens of thousands of bots in a bot network. Typically, botnets consist of a remote command and control (C&C) server that controls the bots in a network and is referred to as the *bot herder* or *bot master*.

A typical bot infection model consists of the following steps.

1. The remote C&C server scans computers in a network to find vulnerabilities. Vulnerability is a bug in software that allows applications to access protected resources and perform privileged actions.
2. The vulnerability is exploited.
3. The bot binary is downloaded and executed in the computer.
4. The bot then establishes communication with the remote C&C server to receive and execute commands to perform malicious actions (denial-of-service, identity theft, spamming, etc.).
5. The bot next seeks to infect computers in other networks.
6. The first two steps are repeated with each computer in the network by the Remote C&C server.

Since, botnets are stealthy; they cause huge damage to victims, whether they be private citizens or business entities. According to the FBI there are “over one million potential victims of botnet cyber crime” [4:1]. Unlike viruses, which act individually, bots receive commands from the remote C&C server and then execute coordinated attacks. A denial-of-service attack is carried out by having the entire set of bots request a page from a web server repeatedly within a short span of time, thereby overwhelming the server. This causes legitimate requests to time out, causing a disruption in service. *Storm worm* [5, 6] is a botnet that made massive parallel network calls to anti-spam websites, such as spameater.com and anti spam groups such as Spamhaus project, thus overloading the servers' capacities and preventing them from responding to requests [7].

Spamming is another malicious activity carried out by bots. First, the remote C&C server sends the spam email to all the bots along with the list of email addresses.

The bots then send the spam email to the email address present in the list. Identification of the actual source of the email spam becomes difficult as the email comes from various sources. Conficker [8] is one such botnet that has a spam capacity of 10 billion per day and has affected approximately three million computers worldwide [9].

Also, botnets have become effective tools for attackers to perpetrate click fraud; in this case, bots are usually plug-ins to popular browsers. The bot runs within the process space of the browser and has access to the document object model of a web page. The bots make HTTP requests to get the page containing the advertisement and mimic a legitimate user by clicking on the advertisement. Again, it is difficult for advertisers to determine whether the click was from a legitimate user or a bot, given that IP addresses are diverse. Clickbot.A [10] is a click fraud botnet used to attack syndicated search engines, and it has infected more than 100, 000 computers.

People give out confidential information, such as their passwords, social security numbers, and credit card data, while performing transactions online and this information is of special interest to attackers. The bot installed in a computer monitors the keystrokes to collect this confidential information and pass it on to the bot master. Since users do not realize that their computers have been compromised, they continue to perform online transactions, while the attacker uses the stolen information to make purchases without the consent of the victims. The FBI prosecuted a person in Los Angeles for using botnets to steal the identities of victims throughout the USA [11].

The first part of this thesis discusses the implementation of a covert botnet model; the second discusses the design and implementation of a defense against such a model.

*1.1 Implementation of a Covert Botnet Model*

A new model for a bot infection [12] has been theoretically proposed. Current detection mechanisms such as BotHunter [13] analyze existing communication patterns of botnets with the Internet to identify a bot infection. The new model differs from existing models in its communication patterns with the Internet. Said communication patterns enable the new model to maintain stealth while at the same time maintain contact with the remote bot master to carry out malicious actions on its behalf. This thesis elaborates on the implementation of the new model and describes the experiments conducted to show that the new model is capable of carrying out malicious actions like those of existing bot infection models. The experiments also demonstrate the proposed model maintains stealth and evades detection from current detection mechanisms such as BotHunter.

*1.2 Design and Implementation of Defense Against Covert Botnet*

A defense mechanism against the new model of bot infection has been designed and implemented. This adds to the existing capability of BotHunter, enabling it to detect the new bot infection model proposed in [12]. The key components of this defense mechanism are:

1. *BotHunter*. This monitors communication with the Internet by analyzing the network traffic entering and leaving the router.
2. *Local Traffic Monitoring System*. This analyzes the traffic entering and leaving network switches. The responsibility of this system is to detect the malicious actions in the local network and communicate them to BotHunter. BotHunter uses the information provided by the local traffic

monitoring system and correlates it with its own analysis to detect the bot infection.

# CHAPTER 2

# RELATED WORK

Initially, internet relay chat (IRC) servers were created for benevolent purposes to allow people having IRC clients to chat together, exchange files, etc.; however, this architecture has been put to malicious use, and IRC servers have been used as remote C&Cs server to control thousands of bots [14, 15]. Attackers use IRC channels to transmit a bot and infect vulnerable computers. The bots that contain IRC client code subsequently communicate with the IRC servers to download commands to carry out malicious actions [14, 16].

*Agobot* [17] is an example of a bot that relies on a centralized remote C&C server and is capable of executing denial-of-service attacks, spamming, and sniffing passwords. The technique to track and investigate botnets is called *botnet tracking* [18].One way of detecting botnets that rely on a central C&C server is to analyze the traffic in known IRC ports for strings that match known commands [1]. Also, techniques to detect Botnet C&C channels in network traffic have also been proposed [19]. Another option is to create a honeypot [20] system consisting of vulnerable systems waiting to be infected by a bot. Once infected, they are used to locate the IRC servers. Bots are then created by system administrators to connect back to the IRC server to profile it.

Some botnets use peer-to-peer networks [21, 22] for communication. They differ from centralized command and control botnets in their network characteristics [17]. Peer-to-peer bots can act as clients and as servers and communicate with other peer bots instead of a central C&C server [17]. Their design is complex, and detection is, consequently, difficult. One of the key advantages of peer-to-peer botnets is resiliency,

as opposed to centralized botnets that fail when the centralized C&C server dies. *Nugache* is one such peer-to-peer botnet. It opens a backdoor on TCP port 8 and is capable of running as a Web server or performs a denial-of-service attack [23]. It has been predicted that peer-to-peer botnets will become more widespread than centralized command and control botnets [17]. The most widespread peer-to-peer bot observed in the wild is *storm worm*. Though detection of peer-to-peer botnets is difficult since there is no central command server, techniques to detect and mitigate peer-to-peer botnets have been proposed [24].

## 2.1 BotHunter

The authors of [12] assume that BotHunter [13] is the best tool for detecting botnets in a local network. BotHunter uses an infection dialog correlation strategy. "In dialog correlation, bot infections are modeled as a set of loosely ordered communication flows that are exchanged between an internal host and one or more external entities" [13:2]. The infection dialog consists of the following five events.

1. *External to Internal Inbound Scan*. The remote command and control server scans computers for vulnerabilities. This is classified as an E1 event by BotHunter.
2. *External to Internal Inbound Exploit*. The bot binary is sent to the vulnerable computer by exploiting the vulnerability. This is classified as an E2 event by BotHunter.
3. *Internal to External Binary Acquisition*. The Bot binary is downloaded by the computer from the remote command and control server by the bot. This is classified as an E3 event by BotHunter.

4. *Internal to External Command and Control Communication*. The Bot downloads commands from the remote command and control server. This is classified as an E4 event by BotHunter.
5. *Internal to External Outbound Infection Scanning*. The Bot tries to infect computers in other networks. This is classified as an E5 event by BotHunter.

BotHunter uses Snort [25, 26] an intrusion detection system to monitor the traffic entering and leaving a network and detect the above events. BotHunter maintains a history of events detected in a computer, and it detects a Bot infection exists in a computer, if either of the following two conditions is met.

*Condition 1.* A computer performs an E2 event followed by an E3, E4, or E5 event.

*Condition 2.* A computer performs at least two of the events E3, E4, or E5.

## 2.2 Covert Botnet Model of Communication

As stated previously, a new model of bot infection dialog has been proposed in [12]. It proposes alternatives to the events mentioned in Section 2.1.

1. A2 Event: Internal to Internal Exploits. The bots local to the network are responsible for infecting other computers in the network.
2. A3 Event: Internal to Internal Binary Acquisition. The bot that downloads the binary from the remote command and control server is responsible for communicating the binary to other bots in the network.
3. A4 Event: Internal to Internal Command and Control Communication. The bot which downloads the command from the remote command and control

server is responsible for communicating the command to other bots in the network.

The model proposed in [12] makes the following assumptions about an initial infection.

1. At least one computer will be infected and obtain the bot binary in some way.
2. The "initial infection avoids detection by some means such as those proposed in [8]" [12:4].

Once the initial infection is completed, the botnet formation proceeds. With the above assumption, the proposed model seeks to evade detection by ensuring that the conditions specified for bot infection detection are not satisfied.

### *2.2.1 Negating Detection Conditions*

*Condition 1.* The bot in the initially infected host tries to infect other computers in the local network. This move is to reduce the number of E2 event from happening. If another computer had been compromised through an E2 event, it will not perform an E3 or E5 event. Instead, it will seek to obtain the binary or commands from other bots within the local network. This prevents Condition 1 from being satisfied.

*Condition 2.* BotHunter maintains the history of events that were detected in a computer and correlates these events to detect a bot infection. So, if BotHunter detects at least two of the E3-E5 events on the same computer, it signals a bot Infection. In the proposed model of bot infection, actions such as bot binary download and command acquisition (E3, E4 events), are not performed by the same bot.

At any point in time, only one bot will communicate with the remote C&C server to perform bot binary download or command acquisition. This bot is the local bot leader, also known as the token bot. The token bot also propagates the binary download or

command acquisition to other bots in the network. This eliminates the need for other bots to contact the remote C&C server. BotHunter does not monitor the traffic that is local to the network and will not see this propagation.

After performing the action, this token bot relinquishes the post of token bot and passes the leadership to another bot in the network. If the previous token bot has done the binary acquisition, the new token bot performs the command acquisition, and if the previous token bot has done the command acquisition, the new bot performs the command binary acquisition. Since the same bot does not perform both the actions, condition two is not satisfied. This enables the botnet to evade detection by BotHunter and maintain its stealth.

### *2.2.2 Peer List Update*

In addition to binary and command acquisition, the bot also requests a list of remote command and control servers. This enables bots to maintain contact with the external network even if one of the remote command and control servers fails. “The Peer list update and command update request would both fall under behavior indicative of internal to external Botnet communication (E4)” [12:7].

### *2.2.3 Information Stored in a Botnet*

Each bot in the network maintains information about its binary version, the actions (E3, E4 events) it has performed, and the timestamp for each action. In addition, bots maintain information about other bots in the network. Information about other bots includes the binary version of a given bot, the actions (E3, E4 events) it has performed, and the timestamp for each. This information is called a report list.

In addition to the above information, the token bot maintains information on the last two external actions performed (E3 or E4) in the botnet (the action history), the action the token bot has performed, and the timestamp of that action. This information is referred to as a token. When a new bot becomes the token bot, the token information is passed to it.

### *2.2.4 Sub-Botnet Management*

For successful operation of the bot network, each of the bots in the network has to coordinate with each other to determine the needed actions, elect a token bot, etc. This is done by exchange of a predefined set of messages among the bots.

### *2.2.5 Token Acquisition Broadcast (TAB)*

When a bot enters the token bot state, it broadcasts a message known as the token Acquisition message which contains the action to be done by this bot. This message also indicates to the other bots that this bot has become the new token bot.

### *2.2.6 Token Report Request Broadcast*

The token bot then sends out a token report request broadcast (TRRB) so that it can get the list of active bots in the network. The token bot compiles a new report list based on the active bots present in the network.

### *2.2.7 Token Report Request Response*

On receiving the TRRB, each bot in the network, replies to the token bot with information about itself. This is called a token report request response (TRRR). The information consists of the MAC address of the host in which the bot resides, the bot

binary version, the action it has performed (if the bot has previously been a token bot), and the timestamp of the action.

### *2.2.8 Token Action Result Propagation*

A token action result propagation (TARP) message is sent after the token bot has performed an action. The TARP is sent to all the other bots in the network and it contains the status of the action performed and the report list compiled after the TRRB. If the bot has successfully downloaded a new bot binary, an "action success" status and the version of the new bot binary is sent in the TARP. If the bot has successfully obtained a new command, an "action success" status is sent in the TARP. On receipt of this message, other bots perform a binary download (A3 event) or command acquisition (A4 event) from the token bot.

### *2.2.9 Token Pass*

After sending the TARP message, the token bot scans the action history to determine which action needs to be performed next and decides on the next token bot based on the following three rules.

1. If there is a bot that has not performed any actions, it is selected as the next token bot; if more than one bot has not performed any actions, one of them is chosen arbitrarily.
2. If there is no bot that has not performed any actions, the token is passed to the bot whose last action matches the next action to be done. If more than one bot matches the action, the one with the oldest timestamp is chosen as the next token bot.

3. If there no such bots that satisfy the above two criterion, the bot with the oldest timestamp, irrespective of the action it performed is selected as the next token bot.

*2.2.10 Token Election*

When a bot initially starts, it sends out a message to check whether a token bot already exists. If there is no response from the token bot, it sends a token election (TE) message. The bot waits for a predefined time interval to receive the report list from other bots. On receipt of this message, the other bots in the network send their report lists. The bot that initiated the token election scans through the report list sent by the other bots and updates its own report list if it is older than the ones received. The bot scans the report list to determine the last two actions performed and then determines the action needing to be performed next. The bot scans the report list again to find the bot whose last action matches the action to be performed next and passes the token to this bot.

*2.2.11 Covert Bot States*

A Bot can be one of in three states:

1. *Token Bot*. Communicates with external peer, and perform actions (E3, E4 events).
2. *Nontoken Bot*. Communicates with token bot to get the results of the latest actions performed by the token bot.
3. *Token Election*. Enters this state when no token bot is present. The purpose is to determine which bot in the network will become the token bot.

Figure 1 illustrates the various states a bot can be at any given time. Table 1 shows the events on which transitions takes place. On startup, the bot checks to see whether a token bot is present. If a token bot is present in the network, the bot transitions to a nontoken bot state. If a token bot is not present, the bot initiates a token election. There are two possible outcomes to a token election for the initiating bot. It can either win the election in which case it moves to a token bot state, it can lose the election to another bot, in which case it move to a nontoken bot state.

A bot in a nontoken bot state can move into the token election state, if a token bot is unresponsive after it sends out a token acquisition broadcast or if it does not send a TARP message. A bot in a token bot state proceeds to a nontoken bot state after a successful Token Pass. A Bot in a nontoken bot state proceeds to a token bot state after receiving a token pass.

### 2.3 Signature-Aware Traffic Monitoring with IPFIX

The authors of [27] propose a signature-aware traffic monitoring system that uses the IPFIX [28] standard. Internet protocol flow information export (IPFIX) is the universal standard for export of flow information to enable network measurement, accounting, and billing [29].

A flow is a set of packets having the same properties, such IP source, destination, protocol, etc., observed in a specific period of time. A metering process located at a router or switch analyzes network traffic and aggregates information into flow records. The exporter process then transmits the flow information from the metering process to the collectors. The data collected from the various exporters is subsequently used for network measurement.

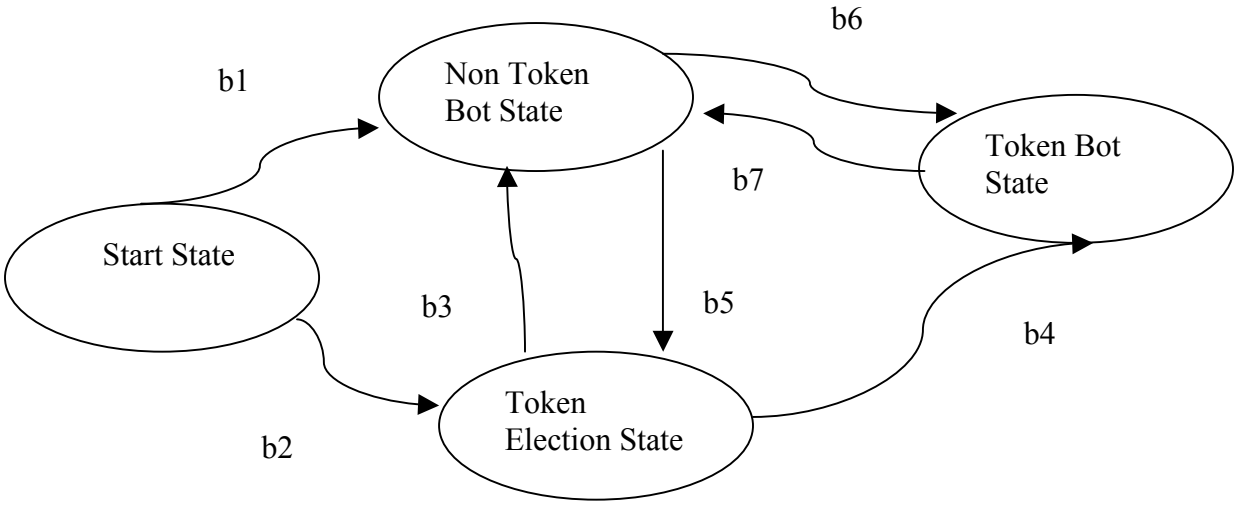


Figure 1. Covert bot states.

Table 1. Events.

| Events | Description |
|---|---|
| b1 | Token Bot present in Botnet |
| b2 | Token Bot absent in Botnet |
| b3 | Bot loses Token Election |
| b4 | Bot wins Token Election |
| b5 | Token Bot is unresponsive |
| b6 | Token Pass received |
| b7 | Token Pass sent |

The monitoring system proposed in [27] consists of signature inspectors that inspect payloads in traffic for signatures and record the signature id of the signature found in the IPFIX compliant flow record. The authors use Snort [25, 26] as their example for signature inspector. The flow record is exported to the IPFIX compliant flow collector, which in turn forwards it to the flow analyzer. The flow analyzer classifies flows with the signature id given in the flow records, to reveal hidden anomaly traffic patterns.

The key differences between the defense outlined in [27] and the design of the defense mechanism for detecting a covert botnet model of bot infection proposed and implemented in this thesis are:

1. The example cited for signature inspector by the authors [27] is Snort which contains a huge database of signatures. Having a signature inspector such as Snort at each monitoring point including switches means replicating the enormous database. Such replication is not necessary for detecting the covert botnet model of bot infection. One centrally located database of signatures and replicating only the detection engine in the monitoring points will suffice. For more details, see Section 4.4.1.

   The disadvantages of having a huge database of signatures at each monitoring points are;

   - Redundancy of the signature database, as the database needs to be replicated at each monitoring point.
   - Increased computational requirements to analyze the traffic with a huge database of signatures.
   - Increased memory to store the database of signatures.
   - Increased time to analyze the traffic.

2. The authors of [27] send the signature id as part of the user data in the flow information. This information is used for flow classification to detect anomaly traffic; however, this classification alone is not useful for detecting the covert botnet model of bot infection. In the proposed model, the dialog correlation strategy proposed by the authors of [13] is extended to include the communication

flows between the internal hosts, in addition to the communication flows that are exchanged between the internal host and external entities. This enables detection of the covert botnet model of bot infection.

# CHAPTER 3

# IMPLEMENTATION OF COVERT BOTNET

This chapter describes the implementation details of the covert bot. In addition, it describes the implementation of other components needed to validate the new covert botnet model. This chapter also outlines the flaw in the new model and presents solutions to solve the flaws. The experimental setup to validate the model is explained. The various experiments performed and the results obtained are discussed in this chapter.

## 3.1 Covert Botnet Framework

The construction of the botnet framework consisted of a programmatic design and implementation of the following components. Table 2 lists the components in the framework.

- *Covert Bot*. This application simulates a bot and is responsible for maintaining contact with external peers to perform binary updates or command updates (E3, E4 events). This communication is illustrated by "Covert Bot 1" downloading a new binary from the External Peer in Figure 2. The bot also propagates the new binary (A3 event) or commands (A4 event) obtained from the external peer to other bots in the botnet.

- *External Peer Application*. This application simulates a remote command and control server and is responsible for sending new versions of the bot binary (E3 event), commands (E4 event) for carrying out malicious actions, and external peer lists (E4 event) to the covert bot.

- *Vulnerable Application*. In the real world, the remote C&C server exploits vulnerability in a computer to perform the infection. Vulnerability in a computer may be software bugs that allow software applications to access protected resources, which can result in an application performing privileged actions like downloading malicious binaries from the Internet and executing them. The vulnerable application implemented for the covert bot framework is a component that simulates vulnerability in the computer. The covert bot exploits this vulnerability to perform the initial infection in the computer (A2 event) as illustrated by "Covert Bot 2" infecting uninfected computer 3 in Figure 2.

### 3.2 External Peer Application

This application is written in C++. It is a TCP socket-based server program and is used to simulate the remote command and control server. It runs on a Linux desktop that is connected to the router, as illustrated in Figure 2. The application listens on a predefined port and depending on the type of request from the Bot, sends the new bot binary, command, or peer list to the requesting bot.

### 3.3 Vulnerable Application

This application is written in C++ and is used to simulate the vulnerability in a computer. It is a TCP socket-based server program and is hosted on all the computers in the local network, as illustrated in Figure 2. The application listens on a predefined port and receives the binary from other bots to create the initial infection (A2 event) in the computer. This application saves the received binary, changes its permission mode to

Table 2. Covert Botnet Components and Actions.

| Components | Actions |
|---|---|
| Covert Bot | Perform A2, A3, A4 events. |
| External Peer Application | Perform E2, E3, E4 events. |
| Vulnerable Application | This Application simulates vulnerability in a computer. Covert Bot exploits this vulnerability to perform the initial infection. |

execute, and assigns it root privileges. The bot binary is then executed by the vulnerable application.

### 3.4 Covert Bot

This application is written in C++. This is the main application that implements the functionality for covert botnet communication in a private subnet. It is a multi-threaded program.

#### *3.4.1 Functions*

The bot performs three important functions. These operations are essential to the working of the botnet.

1. Communicate with external peers (E3, E4 events)
2. Infect other computers in the network (A2 event)
3. Communicate with internal peers (A3,A4 events)

#### *3.4.2 Modules*

The Bot consists of four modules.

1. Network attack module.
2. Message listener module.

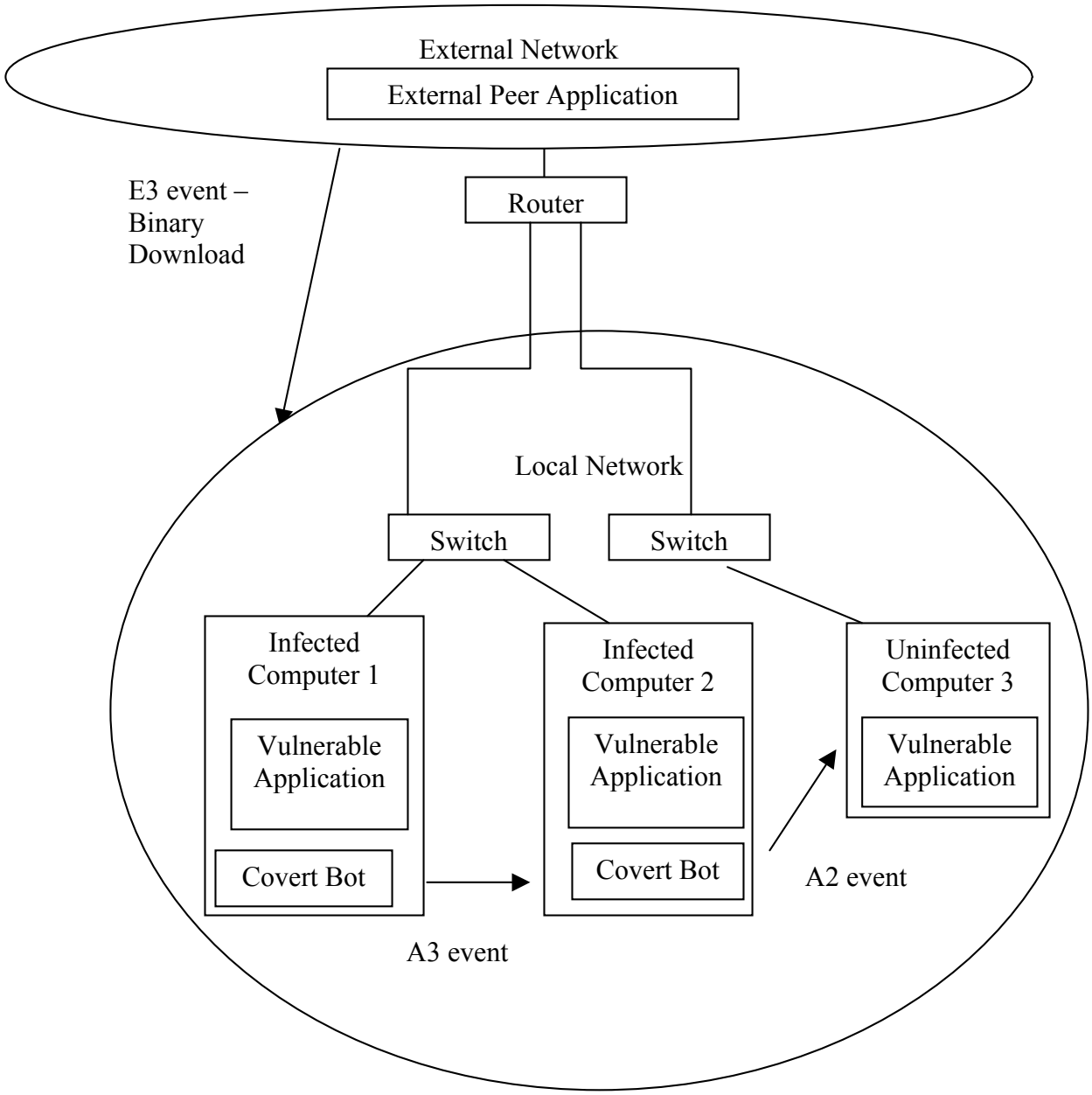


Figure 2. Covert botnet framework.

3. Process module, which is further subdivided into two categories:
    - Token bot module
    - Nontoken bot module.
4. Internal binary update module.

### *3.4.3 Network Attack Module*

The network attack module (NAM) runs on a separate thread. Its main purpose is to perform the initial infection of the other computers in the local network. It sends a special message to check whether a computer is infected. If a bot is present in a computer, the bot indicates that the computer is infected in its response. If there is no response from the computer, the NAM establishes contact with the vulnerable application and sends the bot binary to it (A2 event) to create the initial infection in the computer, as illustrated in the Figure 3.

### *3.4.4 Message Listener Module*

The message listener module runs on a separate thread. This is a raw socket-based server module that listens on a predefined port for messages from other bots in the network. On receipt of any message, it is dispatched to the process module which is responsible for processing the message. The primary idea behind the separation of listening and processing of messages is to ensure that no messages are lost during the processing of messages.

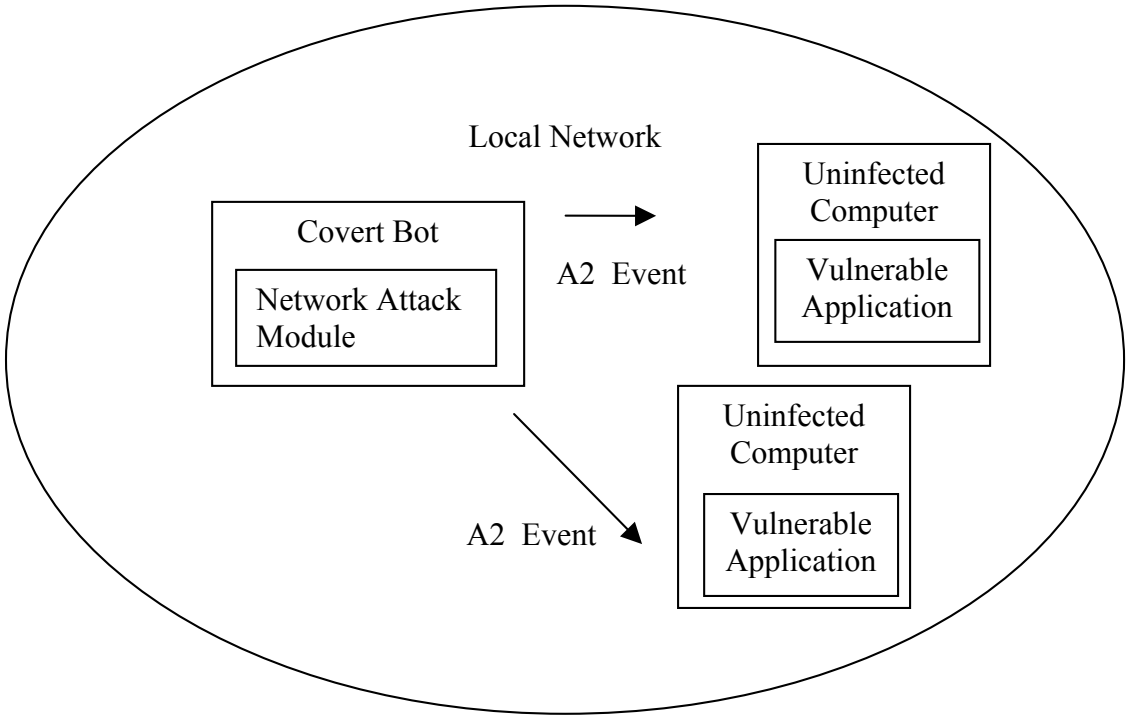


Figure 3. Network attack module.

*3.4.5 Process Module*

The process module runs on a separate thread. It receives messages from the message listener module, and if the bot is running as a token bot, the process module dispatches messages, such as token acquisition broadcast, token report request broadcast (TRRB), token acquisition report, and the Token Pass, to the nontoken bot module.

Figure 4 illustrates the dispatch of messages to the different modules depending on the type of message. The process module is also responsible for conducting the token election. This is performed when the token bot is unresponsive. The TE message is broadcast in the network, and on receipt of this message, other bots in the network respond with their report list.

If the report list of the bot that initiated the TE is older than the report list received from other bots, it updates its report list. The bot waits a predefined time period to receive the report list sent by other bots. After the expiration of the time period, it scans through the report list, to determine the next token bot and passes the token to it.

*3.4.6 Token Bot Module*

This module is responsible for performing actions, such as downloading the new binary, command, or peer list (E3, E4 events), from the external peer application and propagating the result of the action to rest of the bots in the network. As illustrated in Figure 5, the Token Bot module first sends the TAB message to all the bots in the network indicating that it is the new token bot. This action is followed by the TRRB message. On receiving the TRRB, each bot in the network, replies to the token bot with the information about itself. The information consists of the MAC address of the host in which the bot resides, the bot binary version, the actions it performed (if the bot has been a token bot previously), and the timestamp at which the action was performed.

After waiting a fixed period of time and there are no more TRRR responses, this module compiles the new report list. It subsequently performs the action (E3, E4 events) and sends out the status of the action and the new report list as part of the TARP message. After receiving the TARP acknowledgement, it determines the next token bot and performs a token pass to that bot and transfers control to the nontoken bot module.

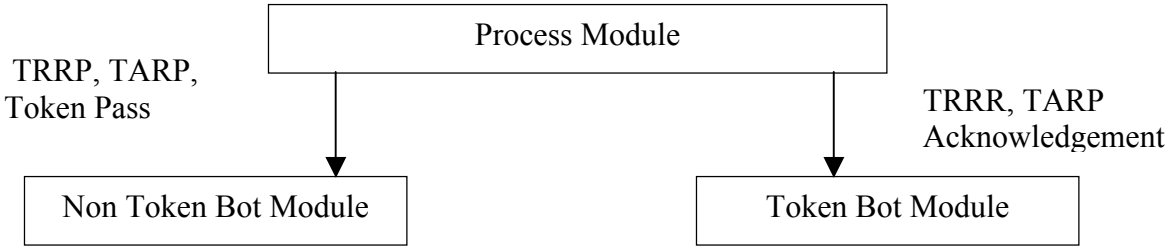


Figure 4. Process module.

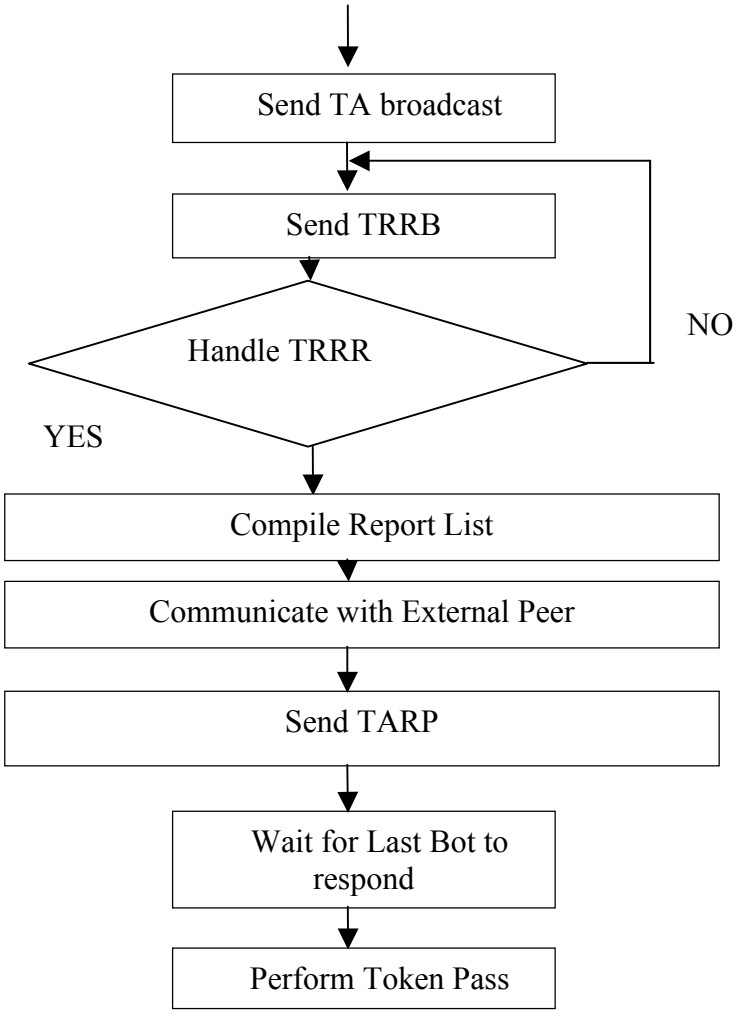


Figure 5. Token bot module flow chart.

*3.4.7 Nontoken Bot Module*

This module is responsible for responding to the TRRB message from the token bot. This module creates the TRRR message which consists of the last action performed by the bot when it was the token bot, the time at which it was performed, and the MAC address of the computer in which the bot resides.

If a new binary has been downloaded by the token bot, this module requests the latest binary from the token bot. It is also responsible for sending the TARP acknowledgement on receipt of the TARP message from the token bot. If the token bot does not send a TRRB or does not respond to the TARP acknowledgement, the nontoken bot module transfers control to the process module which performs the token election. Figure 6 illustrates the program flow in the module.

*3.4.8 Internal Binary Update Module*

This module is responsible for propagating the binary downloaded by the token bot to other bots in the network. If a new binary has been downloaded by the token bot, this is specified in the TARP message that is sent to other bots. On receipt of this information, the bots request the binary from the token bot. The internal binary update module in the token bot sends the new binary to the requesting bot (A3 event). As shown in Figure 7, it can be seen that the Covert Bot 2 requests the latest binary from the token bot, and the internal binary update module sends the new binary to Covert Bot 2.

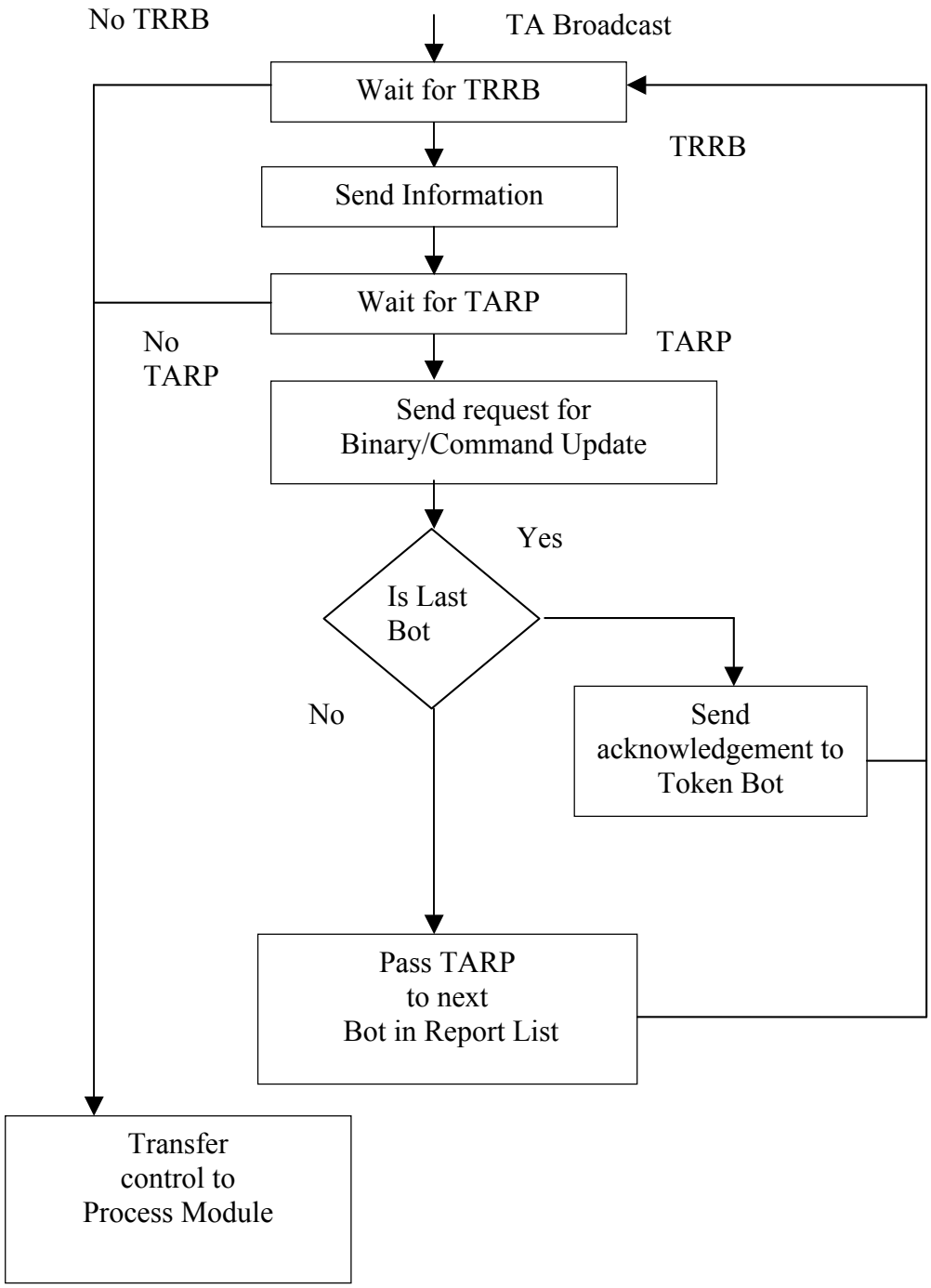


Figure 6. Nontoken bot module flow chart

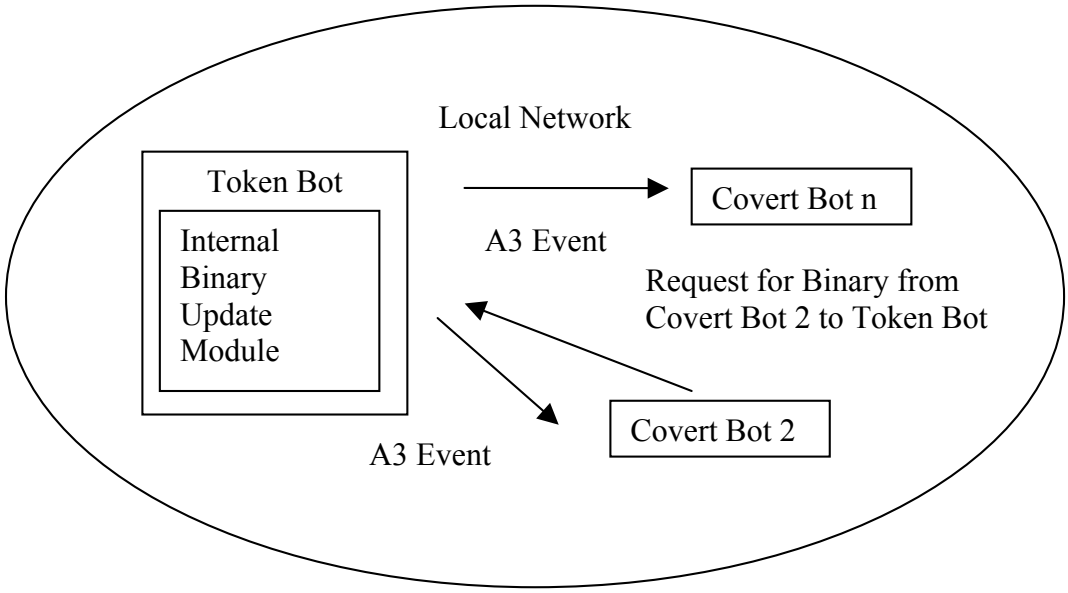


Figure 7. Internal binary update module.

*3.4.9 Messages*

For the successful operation of the bot network, each of the bots in the network has to coordinate with the other bots to determine what action needs to be done, which bot becomes the next token bot, etc. This is done by exchange of a predefined set of messages amongst the bots. The important component of all the messages is the code that indicates the type of message. The code and the message type it identifies are show in Table 3.

*3.4.10 Data Sharing and Synchronization*

This project utilized a thread safe queue to share the data among the various threads. The messages that are received from other bots are put into the thread safe queue. The modules in the bot that are running on various threads wait in the queue, and

Table 3. Message Types.

| Code | Message Type |
|---|---|
| **A** | Token Acquisition Broadcast (TAB) |
| **B** | Token Report Request Broadcast (TRRB) |
| **C** | Token Report Request Response (TRRR) |
| **D** | Token Action Result Propagation (TARP) |
| **E** | Token Pass (TP) |

once a message is received, they are popped out and processed. The covert bot uses the Posix [30] thread libraries for thread creation and synchronization.

Example: A queue is shared between the message listener module and the process module. The message received by the message listener module is pushed into the queue, and a signal is sent to the process module indicating receipt of the new message. On receipt of the signal, the process module pops the message from the queue and processes it.

*3.4.11 Stealth*

The covert bot sniffs on the local traffic to obtain the IP Address and the MAC address from DHCP requests and ARP responses. This helps the covert bot to find new computers to attack in the local network. When the covert bot communicates with other

bots in the network, ARP requests are generated by the switch, if the MAC addresses are not present in the ARP cache in the host and sent to the router. This information can be picked up by an intrusion detection system that monitors the router. In order to prevent the generation of ARP requests, the IP address and the MAC address obtained from sniffing the network traffic are stored in the ARP cache in the host, thus increasing the stealth.

### *3.4.12 TCP*

Communication between the external peer and the internal bots like a binary download and command acquisition (E3, E4 events) is done using TCP. The binary that is propagated (A3 event) to other bots in the local network is also through TCP. This is because the maximum size of data that can be sent thru the MAC layer is 1500 bytes, whereas the typical covert bot binary size is in the region 100 - 500 kb.

### *3.4.13 Persistent Storage*

The bots run in the background on the computer as a Linux service and, hence, do not require manual intervention for startup. Information, such as the last action performed by the bot, the timestamp, binary version of the bot, and the report list, is stored in a file upon bot shutdown as this information is critical in determining the next token bot during a token pass.

If the bot does not store in persistent storage the last action it performed, the bot might perform an action that would satisfy either of the two conditions needed for detection by BotHunter. This is illustrated by the following scenario. Let ‘T’ be the threshold of BotHunter, let four be the size of the Botnet, and assume that none of the

bots have performed any action initially. A recap of the rules for selecting the next token bot is as follows.

1. If there is a bot that has not done any action, it is selected as the next token bot. If there are more than one bot that have not done any action, one of them is chosen arbitrarily.
2. If the first rule is not satisfied, the token is passed to the bot whose last action matches the next action to be done in the botnet.
3. If there are no such bots that satisfy the above two rules, the bot that has the oldest timestamp is selected as the next token bot.

As shown in Table 4, at time t1, Bot 1 performs the binary update. It then selects Bot 2 as the next token bot based on rule 1. On becoming the new token bot, Bot 2 performs the command update, during which time the computer hosting Bot 1 is shutdown. Bot 2 then passes the token to Bot 3 based on rule 1, and after Bot 3 performs its action, it transfers the token to Bot 4.

When Bot 4 performs the TRRB, Bot 1 starts up again. Since Bot 4 does not have persistent storage, it has no way of knowing the last action performed. Hence, it indicates that it has not performed any action in the token report request response message sent to Bot 4. During the token pass, Bot 4 selects a bot based on rule 1. Bot 1 then downloads the new command at t6. Since two events (binary download at t1 and command download at t6) are performed before T, the threshold time of BotHunter, the bot infection is signaled by BotHunter.

However if all Bots maintained persistent storage, then it can be seen from Table 5 that Bot 1 sends the last action performed by it on receipt of the TRRB message from Bot 4. Bot 4 subsequently selects Bot 2 as the next Token Bot as it satisfies rule 2.

Table 4. Scenario 1 – No Persistent Storage.

| **Time** | **Action** |
|---|---|
| t1 | Bot 1 performs Binary Update (E3) and passes token to Bot 2. |
| t2 | Bot 2 performs Command update and passes token to Bot 3 |
| t3 | Computer hosting Bot 1 is shutdown. |
| t4 | Bot 3 performs Peer List Update and passes token to Bot 4. |
| t5 | Bot 4 performs Binary update. Bot 1 starts up and Bot 4 passes token to Bot 1 |
| t6 | Bot 1 performs Command update (E4). Since E3 and E4 are under pruning threshold time T of BotHunter, the Bot infection is identified by BotHunter. |

Table 5. Scenario 2 –Persistent Storage.

| **Time** | **Action** |
|---|---|
| t1 | Bot 1 performs Binary Update and passes token to Bot 2. |
| t2 | Bot 2 performs Command & Control and passes token to Bot 3. |
| t3 | Bot 1 goes down |
| t4 | Bot 3 performs Peer List Update and passes token to Bot 4. |
| t5 | Bot 4 performs Binary update. Bot 1 starts up reads the last action and passes the last action to Bot 4. Bot 4 selects Bot 2 as next Token Bot based on rule 2. |
| t6 | Bot 2 performs Command update. |

## 3.5 Analysis of Existing Botnet Model

### *3.5.1 Experiment Setup*

- *Bot Application*. This application simulated a bot that relies on the existing bot infection model. It was written in C++. It performs binary downloads and external port scan.
- *External Command and Control(C&C) Server*. This application simulated the remote command and control server. When a bot requests a bot binary, the bot binary is sent to it through TCP.
- *Local Network*. The local network was simulated on VMware [31], and the network size was ten. It also consisted of a Linux desktop that hosted BotHunter.
- *External Network*. The external network consisted of two computers. One of them hosted the remote command and control (C&C) server. The other one simulated a computer on which the port scan was performed by the bots.
- *Operating System*. Ubuntu Linux.

Figure 8 illustrates the experimental setup for the experiments. The computer hosting BotHunter was directly connected to the router to analyze the traffic for detecting E3, E4, and E5 events. The Bot in infected computers performed a binary download followed up by a command download or port scan.

### *3.5.2 Experiment 1*

The objective of Experiment 1 was to show that the experimental setup simulates an actual network containing BotHunter and the bots. The binary download and the command download (E3 and E4 events) were performed by the bots within the

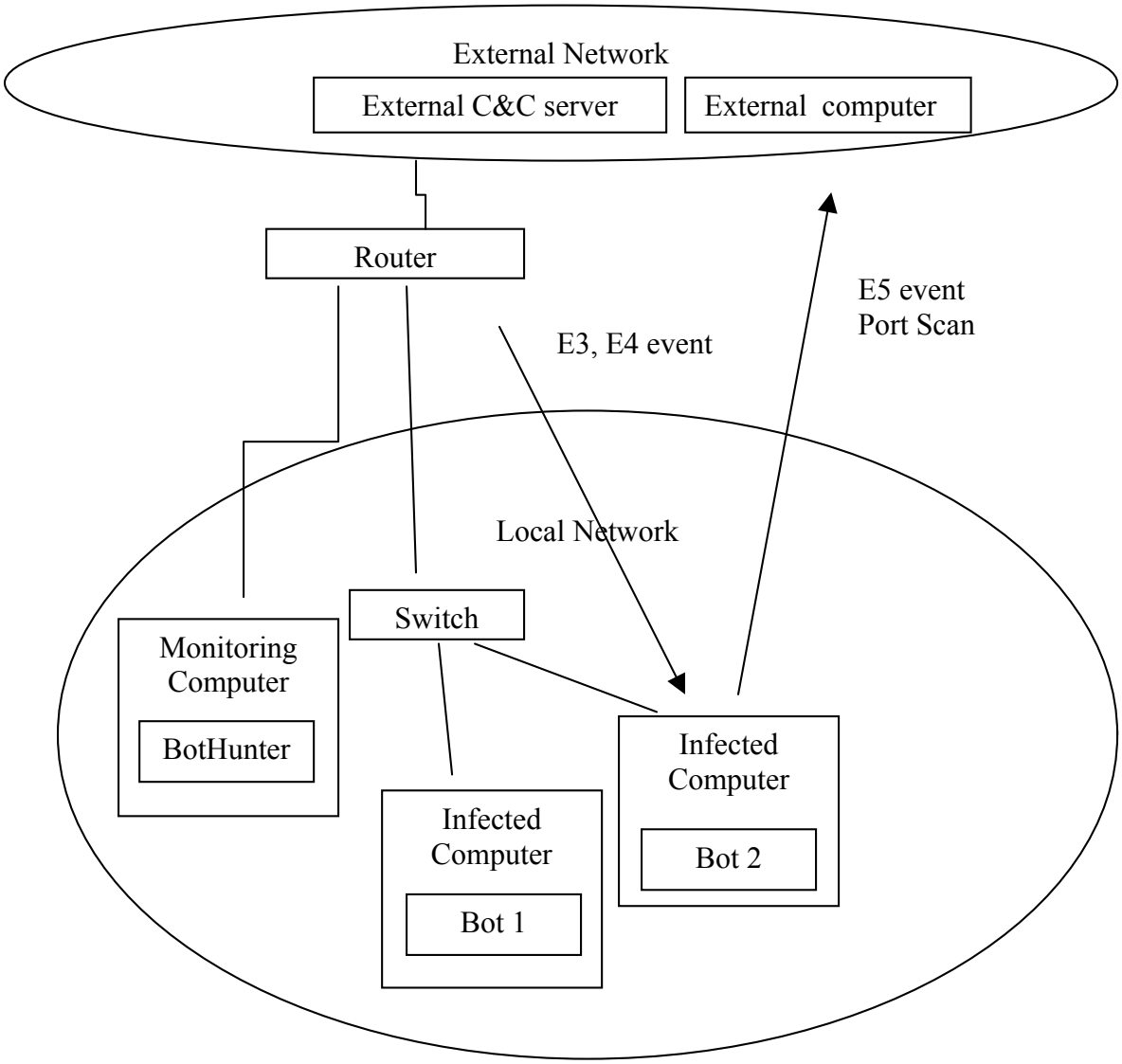


Figure 8. Botnet framework.

threshold time. The BotHunter threshold was set at 5 minutes. The sequence of actions performed by the Bot is shown in Table 6.

The bots were programmed to perform E3 and E4 events every 30 seconds. As seen in Table 6, Bot 1 performed the binary download, followed by the command update within 30 seconds of the binary update. Since the events were conducted within the threshold time period, the botnet was detected by BotHunter.

*3.5.3 Experiment 2*

The objective of Experiment 2 was to show that BotHunter will not flag a bot infection when E3 and E4 events are not done within the threshold time period. The BotHunter threshold time period was set at 5 minutes. The sequence of actions performed by the Bot is shown in Table 7.

The Bots were programmed to perform E3 andE4 events every 5 minutes and 30 seconds. As shown in Table 7, Bot 1 performed the binary download, followed by the

Table 6. BotHunter Detection.

| **Time(MM:SS)** | **Action** | **BotHunter Result** |
|---|---|---|
| 00:00 | Bot 1 performed Binary Download(E3) | |
| 00:30 | Bot 1 performed Command update(E4) | **Detection – Profile generated** |
| 01:00 | Bot 2 performed Binary Download(E3) | |
| 01:30 | Bot 2 performed Command update(E4) | **Detection – Profile generated** |
| 02:00 | Bot 3 performed Binary Download(E3) | |
| 02:30 | Bot 3 performed Command update(E4) | **Detection – Profile generated** |

Table 7. BotHunter Evasion.

| Time(MM:SS) | Action | BotHunter Result |
|---|---|---|
| 0:05 | Bot 1 performs Binary Download(E3) | |
| 5:35 | Bot 1 performs External Port Scan(E5) | **No Detection** since E3, E5 do not occur within the threshold time |
| 11:05 | Bot 2 performs Binary Download(E3) | |
| 16:35 | Bot 2 performs External Port Scan(E5) | **No Detection** since E3, E5 do not occur within the threshold time |
| 22:05 | Bot 3 performs Binary Download(E3) | |
| 27:35 | Bot 3 performs External Port Scan(E5) | **No Detection** since E3, E5 do not occur within the threshold time |

command download after 5 minutes and 30 seconds of the binary download. Since the events were not conducted in the same threshold time period, the Botnet was not detected by BotHunter.

*3.5.4 Experiment 3*

The experiments were conducted for 50 minutes; BotHunter threshold was set at 5 minutes. Figure 9 shows the results of the three experiments, each with a different number of actions per BotHunter threshold time period. The first experiment was conducted with two actions (E3, E4 events) being conducted within the threshold time period, as illustrated in Figure 9. This resulted in detection of bot by BotHunter.

The second experiment was conducted with the bot performing only one action per threshold time. Ten actions were performed by the bot within the 50 minutes. The third experiment was conducted with one action per twice the threshold time period. This

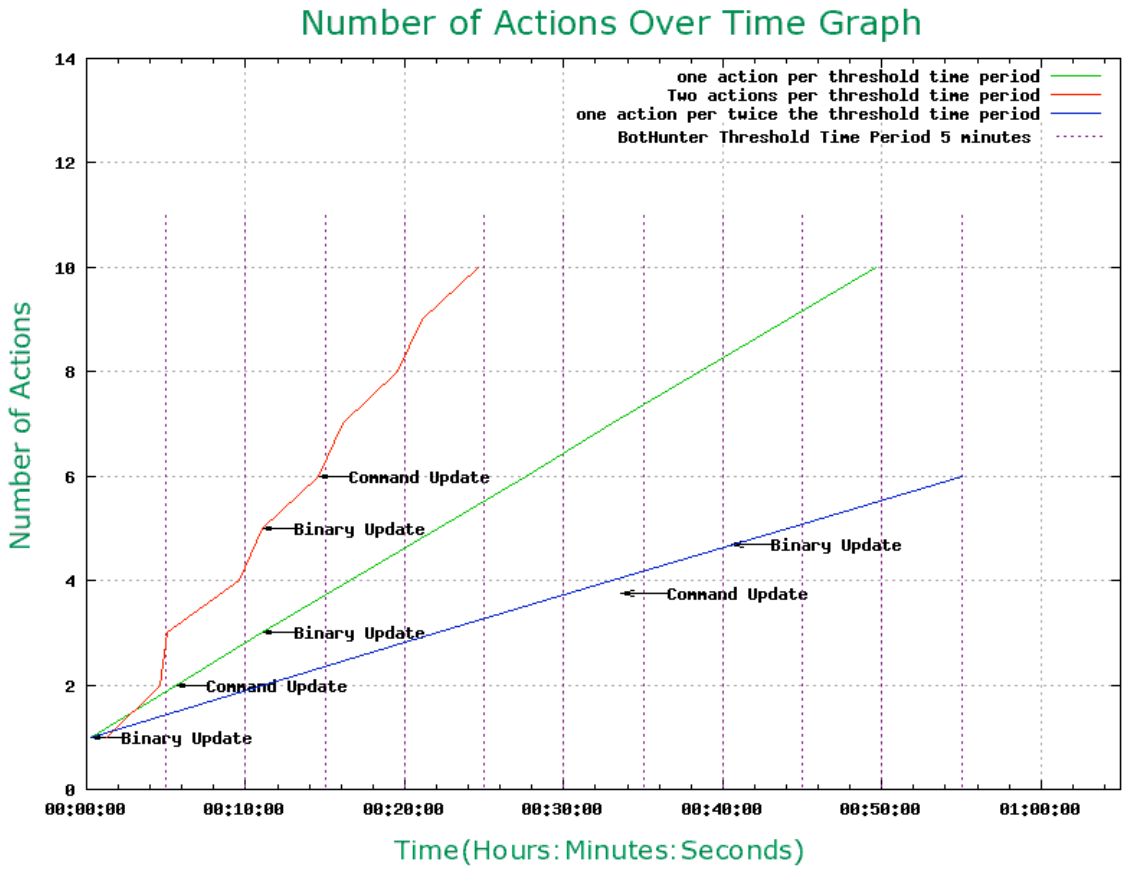


Figure 9. Number of actions over time graph.

resulted in no detection by BotHunter, but the number of actions was reduced to five during the same 50-minute time period.

## 3.6 Analysis of Covert Botnet

### *3.6.1 Experiment Setup*

- *Covert Bot Application*. This application simulated the covert bot. It was written in C++. It was responsible for performing the binary, command, and peer list download (E3, E4 events) from the external peer and propagating

these actions (A3, A4 events) to other covert bots in the local network. It was also responsible for performing the initial infection (A2 event) in other computers in the local network.

- *External Peer Application*. This application simulated the remote command and control server. When a bot requests a bot binary, the bot binary is sent through TCP.
- *Local Network*. The local network was simulated on VMware, and the network size was ten. It also consisted of a desktop computer that hosted BotHunter.
- *External Network*. The external network consisted of one desktop computer, which hosted the external peer application.
- *Operating System*. Ubuntu Linux.

Figure 10 illustrates the experimental setup; all the infected computers hosted the covert bot. The token bot downloaded the binary or command (E3, E4 events) from the external peer application and propagated (A3, A4 events) it to the rest of the bots in the local network. The BotHunter running a Linux desktop could trap the E3, E4 events as it analyzed communication with the external network; however it could not identify the A3, A4 events as it did not monitor the traffic within the local network.

### *3.6.2 Experiment*

The objective of this experiment was to show that under the covert bot model, BotHunter will not flag a bot infection even when E3 and E4 events are done within the threshold time. The BotHunter threshold was set at 5 minutes. The covert bot

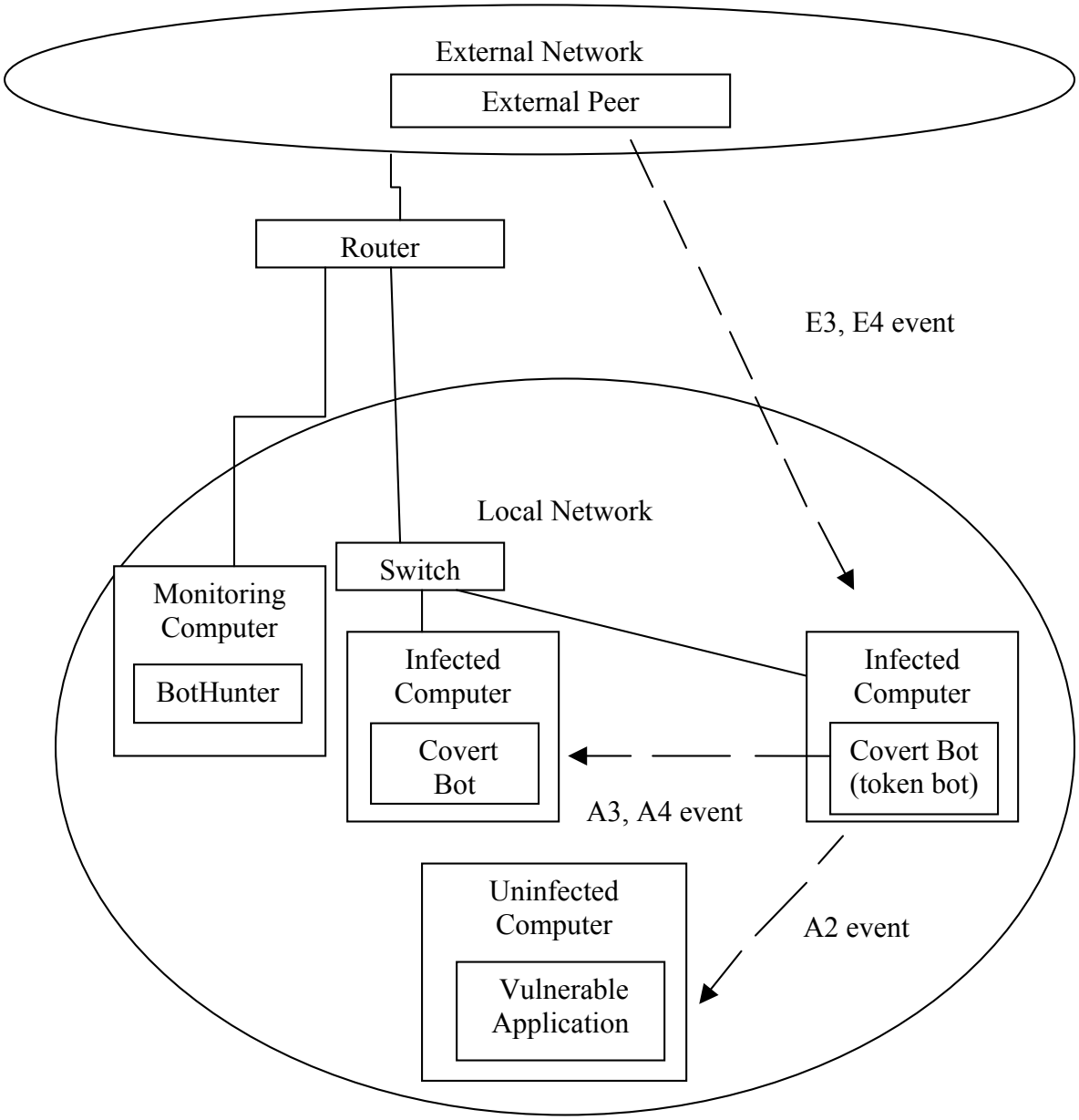


Figure 10. Covert botnet framework.

was programmed to perform its action (E3, E4 event), followed by internal propagation (A3, A4 event) ,and a token pass event every 1 minute and 30 seconds. The sequence of actions performed by the bots is shown in Table 8.

As illustrated in Table 8, Bot 1 performed the binary download (E3 event) at 1 minute and 30 seconds. This latest binary was immediately propagated (A3 event) to the other bots in the local network. Bot 1 next performed a token pass to Bot 2. Similarly, Bot 2 performed the command download (E4 event) at 3 minutes, which was immediately propagated (A4 event) to the other bots in the local network. It then did a token pass to Bot 3. Even though the E3, E4 event was performed within the BotHunter's

Table 8. Covert Botnet.

| **Time (MM:SS)** | **Token Bot** | **Action** | **Other Bots** | **Token Pass** |
|---|---|---|---|---|
| 01:30 | Bot 1 | Binary Download | A3 event | Bot 2 |
| 03:00 | Bot 2 | Command Download | A4 event | Bot 3 |
| 04:30 | Bot 3 | Peer List Download | A4 event | Bot 4 |
| 06:00 | Bot 4 | Binary Download | A3 event | Bot 5 |
| 07:30 | Bot 5 | Command Download | A4 event | Bot 6 |
| 09:00 | Bot 6 | Peer List Download | A4 event | Bot 7 |
| 10:30 | Bot 7 | Binary Download | A3 event | Bot 8 |
| 12:00 | Bot 8 | Command Download | A4 event | Bot 9 |
| 13:30 | Bot 9 | Peer List Download | A4 event | Bot 10 |
| 15:00 | Bot 10 | Binary Download | A3 event | Bot 2 |
| 16:30 | Bot 2 | Command Download | A4 event | Bot 3 |
| 18:00 | Bot 3 | Peer List Download | A4 event | |

threshold time, they were performed by different bots. Further, since infection dialog in BotHunter is tied to a single computer, this model evades detection by BotHunter.

*3.6.3 Number of Actions Graph*

Figure 11 shows the results of the two experiments, each with a different number of actions per BotHunter threshold time period. The BotHunter threshold time period was set at 5 minutes. In the first experiment, the bots were programmed to perform three actions (binary download, command download, peer list download) at one-and-a-half minute intervals. The three actions are shown in Figure 11 during the time interval between 5 and 10 minutes. Within 18 minutes, 12 total actions were performed by the bots. In the second experiment, the bots were programmed to perform actions (E3, E4 events) at intervals lower than one-and-a-half minutes, which resulted in 12 actions being performed in less time.

The experiments show that in the covert botnet model, the bots could be programmed to do any number of actions within the threshold time period as the model negates the two conditions required for detection as discussed in Section 2.2.1. The only drawback is the increase in network bandwidth because the increase in actions generates increased traffic.

## 3.7 Comparison of Number of Bot Actions over Time

*3.7.1 Existing Botnet Model*

In order to evade detection, bot actions have to be spaced out by BotHunter pruning threshold Time ‘T’. The BotHunter threshold time period was set at five minutes.

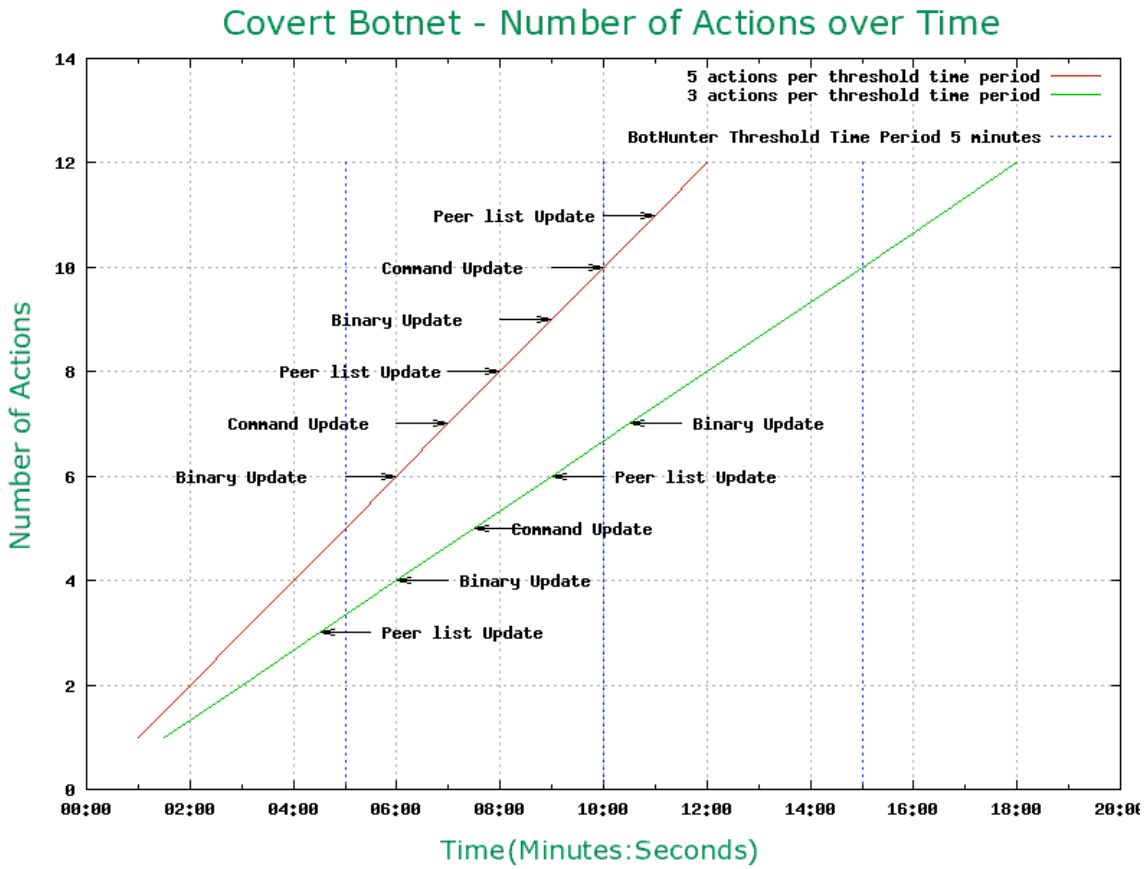


Figure 11. Covert botnet - Number of actions over time.

It takes about an hour to perform 12 actions, as shown in Figure 12.

*3.7.2 Covert Botnet Model*

The bots were programmed to do three actions per threshold time period, and it took 18 minutes to complete the 12 actions, as illustrated in Figure 12. Unlike the existing botnet model wherein only one action can be done within the BotHunter threshold time period in order to evade detection, the covert botnet model can do any number of actions within the threshold time. Consequently, the bots could be

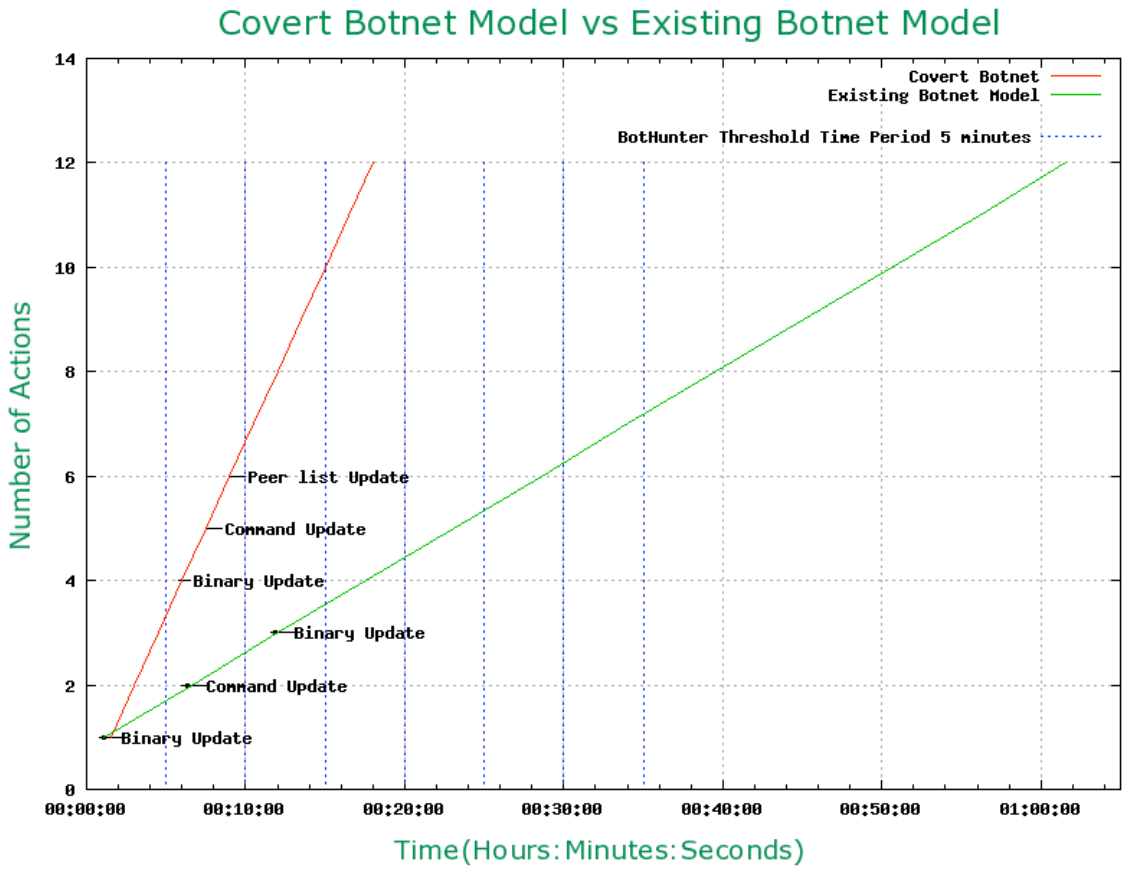


Figure 12. Comparison of number of bot actions over time.

programmed to perform more actions within the threshold time period, in which case the time taken to complete 12 actions would be much less.

## 3.8 Comparison of External Network Traffic

### *3.8.1 Existing Botnet Model*

If the size of the binary downloaded is S KBs, and the number of bots in the network is N, the amount of traffic generated between the internal host and the external peer is N*S KBs. This is because each bot has to download the bot binary from the

external peer. As can be seen in Figure 13, when the botnet size is 10 and the size of the binary download is 165 KB, the total data downloaded from the external peer is 1650 KB. An increase in traffic could lead to possible detection by intrusion detection systems that monitor the traffic entering a local network.

*3.8.2 Covert Botnet*

In the case of the covert botnet model, the amount of traffic generated between the internal host and the external peer is a constant S. Since only one bot is going to download the bot binary and distribute it to the other bots within the network. As can be seen in Figure 13, irrespective of the size of the botnet, the traffic between the internal host and the external peer is 165 KB which is the size of the bot binary.

**3.9 Covert Botnet Connectivity**

In a network, computers are shutdown either because of human intervention or because of extraneous circumstances like a power failure. A computer that is shutdown could contain a covert bot which is either in a token bot state or a nontoken bot state. When the token bot is not alive, the connectivity with the external peer is lost, leading to a breakdown in external communication. A loss of external communication means that the bots do not have access to the latest binary and are not be able to perform malicious activities, leading to a collapse of the bot network. To deal with this, the token election process had been proposed in the covert botnet model.

Experiments were conducted to determine how effectively the token election process restores the connectivity of the botnet when computers are shutdown and restarted. The experiment was simulated on a VMware virtual network consisting of 30

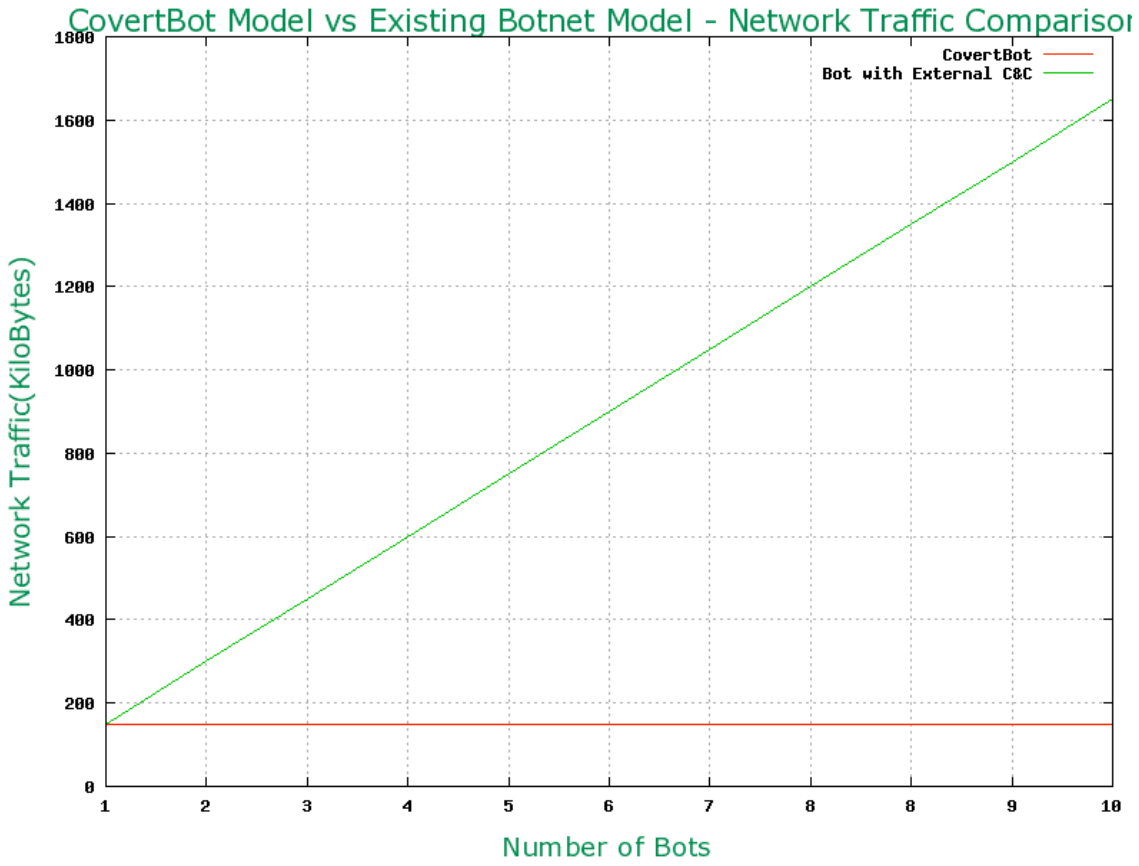


Figure 13. Comparison of external network traffic.

computers. The token pass was programmed to occur every 2 minutes. Actions such as binary and command downloads were performed every 2 minutes.

In order to simulate computers containing a token bot shutting down, the bot in a token bot state was programmed to generate a random number before performing a token pass. If the number was within a threshold value, the bot did not respond to any message from other bots in order to simulate an unresponsive bot. Upon not receiving any responses from the token bot, the other bots would initiate a token election. Figure 14 illustrates the experimental setup. Each of the 30 computers in the setup contained a covert bot.

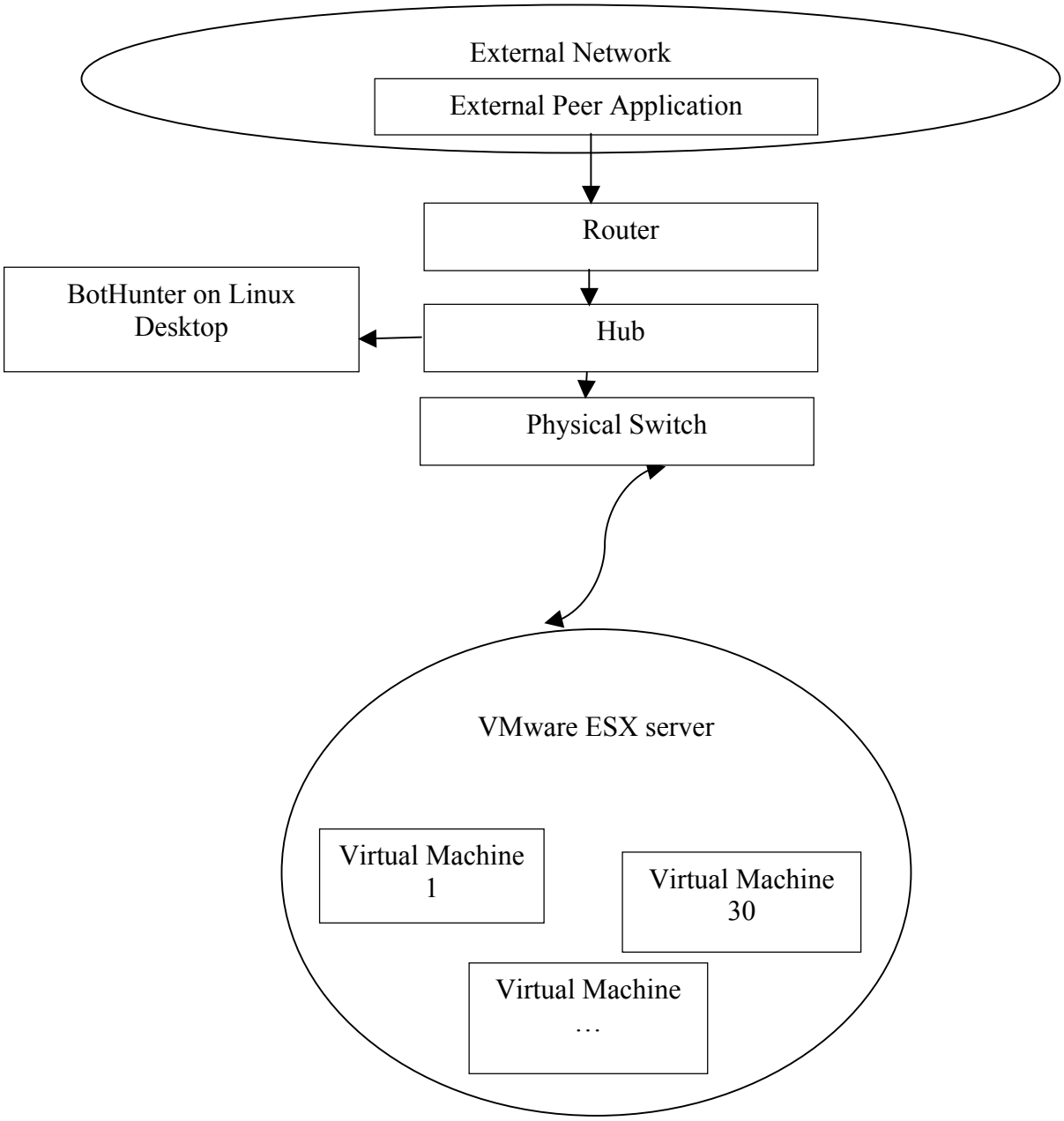


Figure 14. Experiment setup.

*3.9.1 Covert Botnet Connectivity – Token Bot*

The graph in Figure 15 below shows the connectivity of the bot network with the external peer over time. The botnet is disconnected from the external peer when the covert bot that is in token bot state becomes unresponsive. It is during said time that the token election is conducted. On an average, it took about 5 minutes for a token election to elect the new token bot and restore connectivity with the external peer.

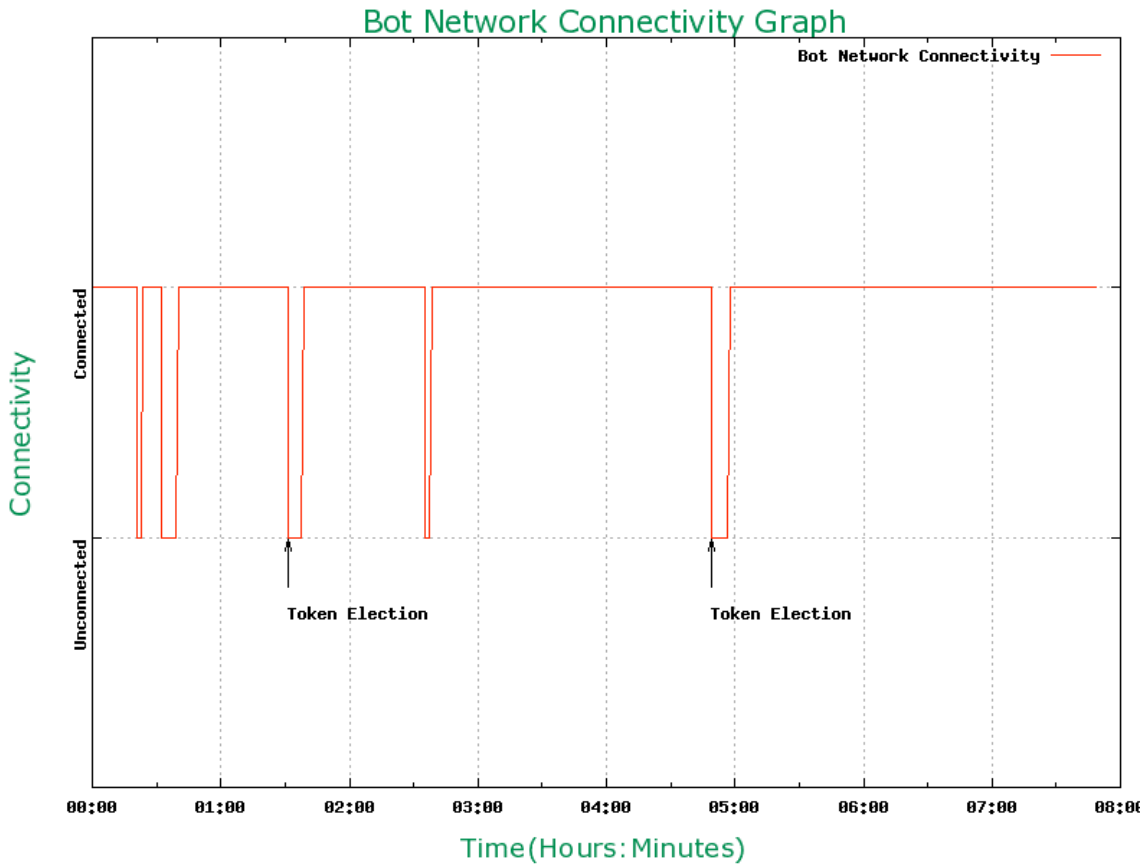


Figure 15. Covert botnet connectivity – Token bot.

The graph in Figure 16 below shows the number of actions performed over time when bots randomly become unresponsive. The number of actions over time increased linearly. Further, when a covert bot that was in a token bot state was unresponsive, there was a temporary plateau due to a token election. The number of actions again increased when a bot was elected as a token bot.

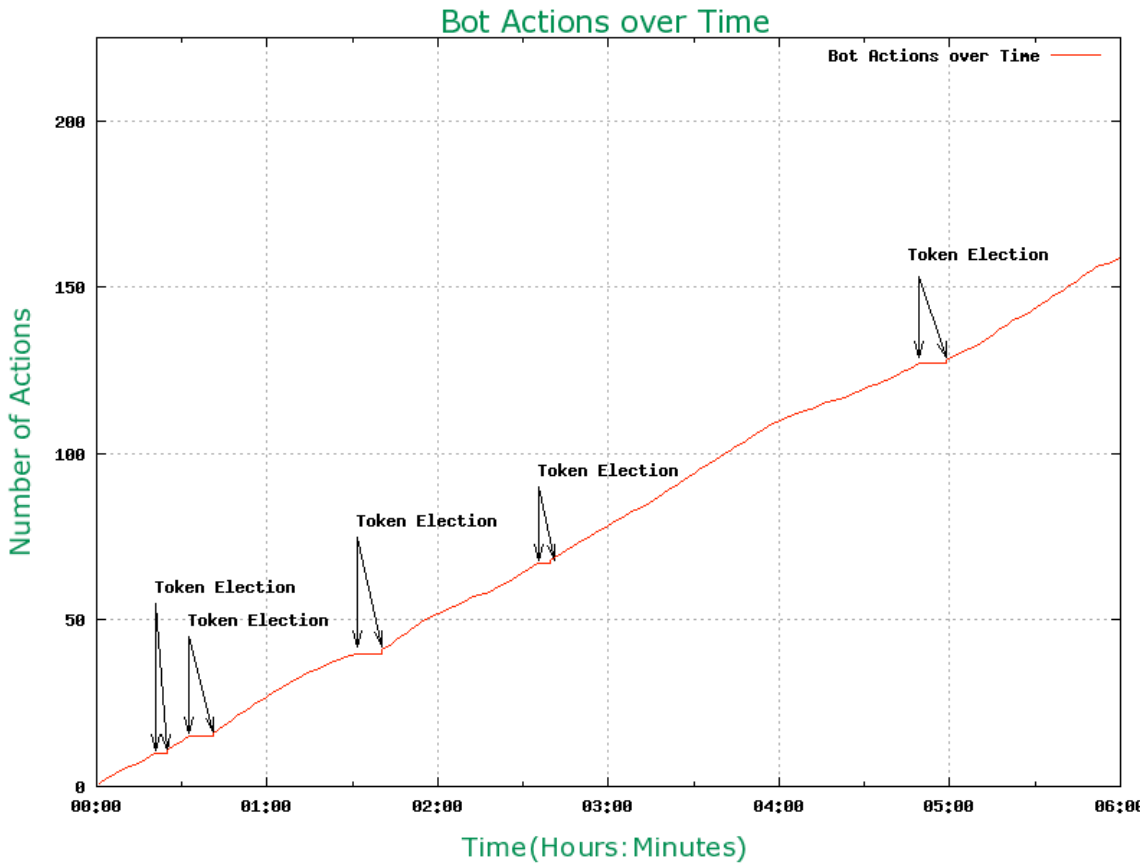


Figure 16. Covert bot actions over time with token election.

*3.9.2 Covert Botnet Connectivity - Nontoken Bot*

The graph in Figure 17 below shows the connectivity a nontoken bot has with the botnet. When a bot in nontoken bot state is unresponsive, its connectivity with the bot network is lost. When the bot resumes, it sends out a special message to determine if a token bot is present in the botnet. If a token bot is present, it responds to the message, and the connectivity of the bot is instantly established.

If the bot resumes during a token election, the bot has to participate in the token election process, and the connectivity of the bot is established when the token election is complete.

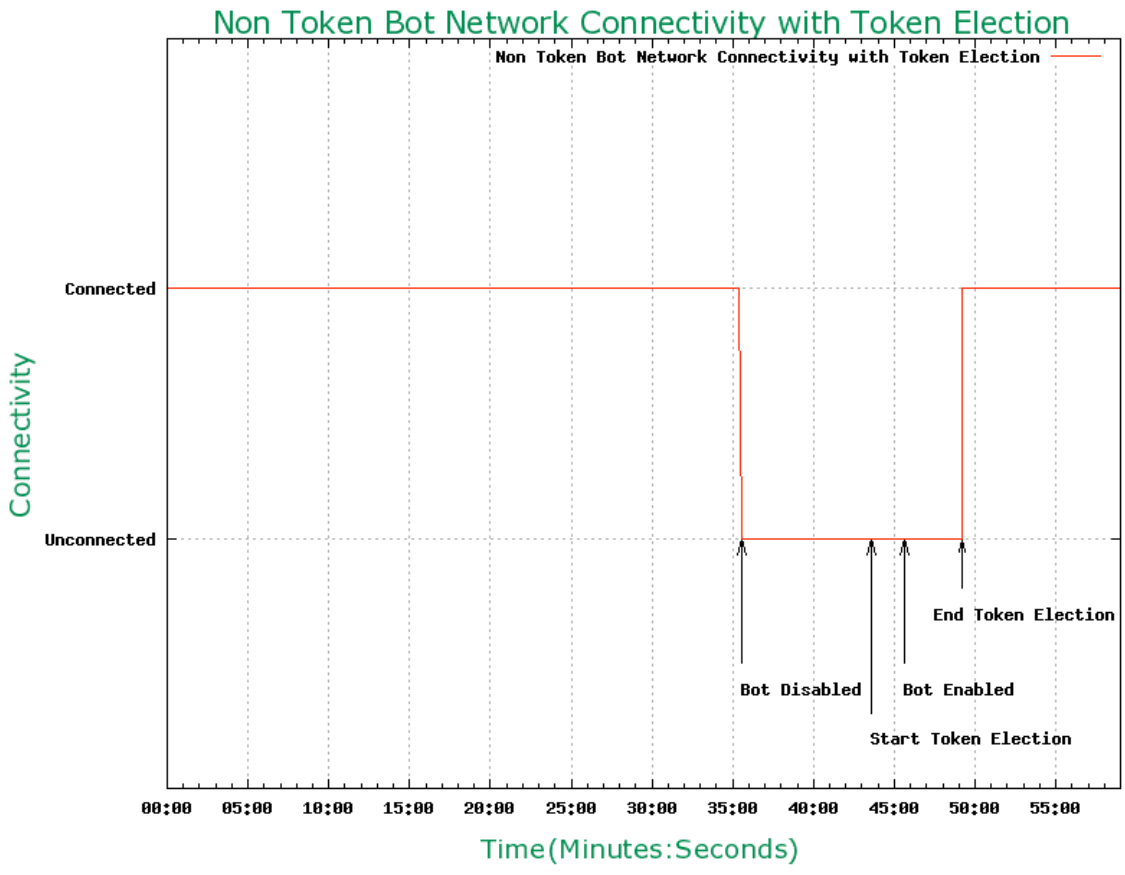


Figure 17. Nontoken bot network connectivity with token election.

## 3.10 Covert Botnet Flaws

### *3.10.1 Token Pass Flaw*

The design for BotHunter evasion had a flaw that needed to be rectified. A recap of the rules for selecting the next token bot follows.

1. If there is a bot that has not done any action, it is selected as the next token bot. If more than one bots exist that has not done any action, one of them is chosen arbitrarily.
2. If the first rule is not satisfied, the token is passed to the bot whose last action matches the next action to be done in the botnet.
3. If there are no such bots that satisfy the above two rules, the bot that has the oldest timestamp is selected as the next token bot.

The flaw is in rule 3.The following scenario illustrates the flaw. Let the botnet size be four and the BotHunter threshold be T. As shown in Table 9, initially Bot 1

Table 9. Token Pass Flaw.

| **Time** | **Token Bot** | **Action** | **Bot Status** | | | **Bot Hunter Result** |
|---|---|---|---|---|---|---|
| t1 | Bot 1 | Binary update | Bot 1 | Bot 2 | Bot 3 | |
| | | | Alive | Alive | Alive | |
| t2 | Bot 2 | Command update | Bot 1 | Bot 3 | Bot 4 | |
| | | | Alive | Alive | Alive | |
| t3 | Bot 3 | Peer List Update | Bot 1 | Bot 2 | Bot 4 | |
| | | | Alive | Alive | Alive | |
| t4 | Bot 4 | Binary Update | Bot 1 | Bot 2 | Bot 3 | |
| | | | Alive | **Dead** | Alive | |
| t5 | Bot 1 | Command Update | Bot 2 | Bot 1 | Bot 3 | Detection/Profile generated |

becomes the token bot and performs binary update; next Bot 2 becomes the token bot and performs command update.

When Bot 4 becomes the token bot, the computer hosting Bot 2 is shutdown. After completing the binary download, Bot 4 selects Bot 2 as the next token bot, since its last action matches the next action to be performed, e.g., command update. However, Bot 2 is unresponsive. Subsequently, Bot 4 selects Bot 1 as the next Token Bot as it satisfies rule 3 of token bot selection. Bot 1 completes its action at t5; however, Bot 1 has performed two actions triggering an E4 and E5 event within the BotHunter threshold 'T' time period leading to detection by BotHunter.

### *3.10.2 Solution to Token Pass Flaw*

The flaw could be rectified by excluding rule 3 for selection of the next token bot. During a token pass, if there are no bots which satisfies rule 1 and rule 2, the token bot will wait until such time a bot that satisfies rule 1 or rule 2 comes into existence.

### *3.10.3 Timestamp Flaw*

In the current design, each bot stores the time at which it performed an action (E3, E4 event). This time is based on the time in the local computer. It could be possible that the times in some computers in a network might differ. If such be the case, a problem during the token acquisition response propagation (TARP) could result. When the TARP message is being passed around, each bot checks to see if the timestamp in the report list in the TARP message is greater than the timestamp of the report list it is currently storing. If it is greater, it is accepted; otherwise, it is rejected.

The problem with using timestamps for report lists is illustrated by the following scenario, in which three computers have the same time, while the fourth one is behind the other three by t minutes. As shown in Table 10, Bot 1 completes its action at t1, Bot 2 completes its action at t2, Bot 3 completes its action at t3, and Bot 4 completes its action at t4.

After completing its action, Bot 1 passes the report list as part of the TARP message, and the other bots accept the message as the timestamp (t1) that is greater than the initial timestamp (0) stored in them. Similarly, Bot 3 passes the report list as part of the TARP message, and the other bots accept the message as the timestamp (t3) is greater than the timestamp (t2) stored in them.

Table 10 Timestamp Flaw.

| **Time** | **Token Bot** | **Action** | **TARP Report List Timestamp** | | | **Token Pass** |
|---|---|---|---|---|---|---|
| t1 | Bot 1 | Binary Download | Bot 2 | Bot 3 | Bot 4 | Bot 1 passes token to Bot 2 |
| | | | t1 | t1 | t1 | |
| t2 | Bot 2 | Command Update | Bot 1 | Bot 3 | Bot 4 | Bot 2 passes token to Bot 3 |
| | | | t2 | t2 | t2 | |
| t3 | Bot 3 | Peer List Update | Bot 1 | Bot 2 | Bot 4 | Bot 3 passes token to Bot 4 |
| | | | t3 | t3 | t3 | |
| t4 | Bot 4 | Binary Download | Bot 1 rejects the Report List since t4<t3 | Bot 2 rejects the Report List since | Bot 3 rejects the Report List since | No Token Pass |

However, since the computer hosting Bot 4 is running behind the other infected computers, the timestamp (t4) of Bot 4 is lesser than the timestamp (t3) of the report list in the other bots. Hence, the TARP message is rejected by the other bots. Since all the bots have rejected the TARP message, the token bot will not receive the TARP acknowledgement from other bots and hence will not perform a token pass, leading to a collapse of the botnet.

### *3.10.4 Solution to Timestamp Flaw*

The above problem could be avoided by eliminating use of the timestamp to determine the latest report list. The alternative is to use an action counter to determine the latest report list. The action counter indicates the total number of actions performed by the botnet. The solution is illustrated by the following scenario, in which three computers have the same time, while the fourth one is behind the other three by t minutes. As shown in Table 11, initially Bot 1, which is in the token bot state, performs the binary download and increments the action counter to 1.

Table 11. Action Counter Solution.

| **Time** | **Token Bot** | **Action Counter** | **Action** | **Bot Number - action counter** | | | **Token Pass** |
|---|---|---|---|---|---|---|---|
| t1 | Bot 1 | 1 | Binary | Bot 2 - 1 | Bot 3 – 1 | Bot 4 - 1 | Bot 2 |
| t2 | Bot 2 | 2 | Command | Bot 1 - 2 | Bot 3 -2 | Bot 4 - 2 | Bot 3 |
| t3 | Bot 3 | 3 | Peer list | Bot 1 - 3 | Bot 2 - 3 | Bot 4 - 3 | Bot 4 |
| t4 | Bot 4 | 4 | Binary | Bot 1 - 4 | Bot 2 - 4 | Bot 3 - 4 | Bot 2 |

It then propagates the action counter as part of the TARP to the other bots. Similarly, Bot 2 performs its action and increments the action counter. When Bot 4 finally becomes the token bot, it performs its action, increments the action counter, and passes it on to rest of the botnet. The TARP is accepted by the other bots even though Bot 4 is behind in time. The Botnet operation continues unhindered.

# CHAPTER 4

# IMPLEMENTATION OF DEFENSE AGAINST COVERT BOTNET

This chapter presents the concept for the defense against the covert botnet proposed in [12]. It explains the implementation details of the key components in the defense system, the experimental set up, the experiments conducted, and the results obtained to validate the concept.

## 4.1 Overview of Defense System

BotHunter monitors the traffic entering or leaving a network and detects all E3 through E5 events; however, it does not monitor the traffic within the local network and, hence, does not detect an internal to internal exploit (A2 event), internal to internal binary acquisition (A3 event), to internal command and control communication (A4 event). In order to detect these events, this thesis proposes a local traffic monitoring system (LTMS).

## 4.2 Rules for Detection

In order to declare the existence of a Bot infection either one the following conditions has to be satisfied.

- An internal to external binary acquisition (E3) event that is followed by an A2 event within the BotHunter threshold time period.
- An internal to external binary acquisition (E3) event that is followed by an A3 event within the BotHunter threshold time period.

- An internal to external C&C communication (E4) event that is followed by an A4 event within the BotHunter threshold time period.

## 4.3 Components of Defense System

The key components of the Defense are the 1) Local Traffic Monitoring System and 2) the BotHunter.

## 4.4 Local Traffic Monitoring System

The local traffic monitoring system (LTMS) analyzes the traffic flowing through the network switch to detect and report A2, A3, and A4 events. The three main components of LTMS are the internal event detector, internal event dispatcher, and central event aggregator. There may be one or more internal event detectors and internal event dispatchers in the system. They exchange information with the centrally located event aggregator, which in turn liaisons with BotHunter.

### *4.4.1 Internal Event Detectors*

The internal event detectors analyze the traffic local within the network for signatures. A signature is a raw sequence of bytes or strings. These raw sequences of bytes or strings are present in the bot binary or commands. As shown in Figure 18, this component may be an application running on a computer that gains access to local traffic by connecting to a network switch configured for port mirroring or it may be an application that is resident in the network switch itself.

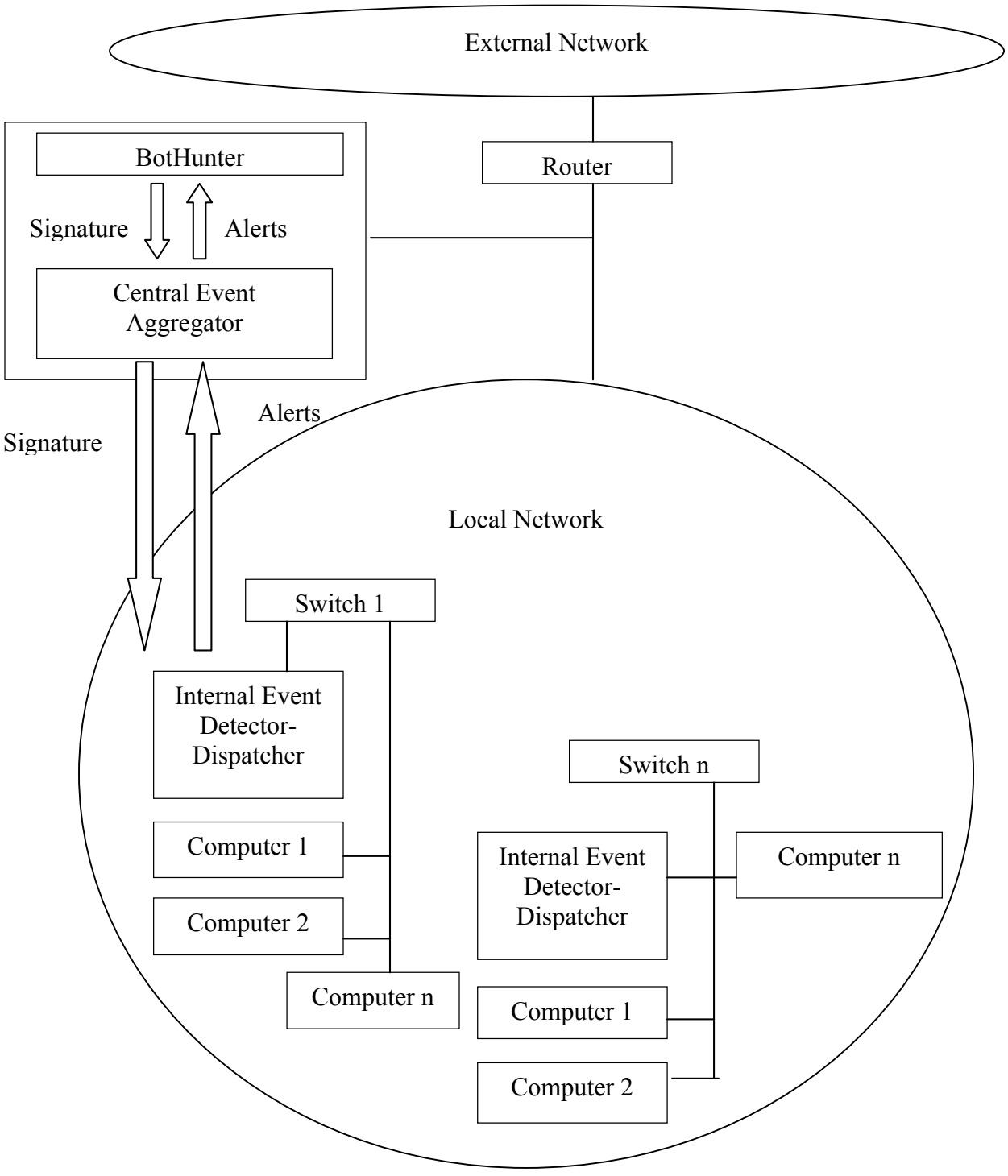


Figure 18. Defense setup.

This component does not maintain a database of signatures but instead relies on the signature database that is available in BotHunter. BotHunter internally uses Snort, which has an extensive database of signatures. Snort produces an alert message when an E3 through E4 event is detected. This alert message consists of the signature that was found in the network traffic from the remote command and control center.

Since, the same binary or command that was downloaded during an E3 through E4 event is propagated across the local network by the covert bot, the internal event detector utilizes the signature contained in the alert message generated by Snort in BotHunter for analyzing the local traffic. Unlike the signature inspector proposed in [27] in the signature-aware traffic monitoring with IPFIX, this component does not need a database of signatures. This results in four benefits.

- *Memory*. Since it stores only the recent signature sent by BotHunter it requires far less memory than it would if has to have the entire database of signatures.
- *Computational Load*. The computational requirements are greatly reduced since we need to search for only one signature.
- *Time*. The time needed to analyze the network traffic is reduced.
- *Speed*. The speed at which network traffic can be analyzed is increased.

Once the A3 through A4 event is detected by this component, the time of detection, the type of event (A3, A4), the internet protocol address of the computers participating in the events, and the signature id are sent to the internal event dispatcher.

*4.4.2 Internal Event Dispatcher*

This component formats the information related to an A3 through A4 event provided by the internal event detector in the form of a Snort alert message. This alert message is then passed on to the central event aggregator. This component is also responsible for receiving the signature from the central event aggregator and passing it on to the internal event detector.

*4.4.3 Central Event Aggregator*

This component receives the signature from BotHunter that was used in detecting the E3 through E4 event. It passes the signature on to one or more Internal Event Dispatchers. It also forwards alerts from one or more internal event dispatchers to BotHunter.

**4.5 BotHunter**

BotHunter sends the signature that was used in identifying the E3 through E4 events to the central event aggregator. It correlates the external events with the internal events provided by the central event aggregator and signals a bot infection if it matches one of the rules for detection as outlined in Section 4.2.

**4.6 Implementation of Local Traffic Monitoring System**

Instead of building a proprietary protocol for the exchange of information between the components in the LTMS, this paper proposes using the Internet protocol flow information export (IPFIX) protocol. IPFIX is the universal standard for export of flow information to enable network measurement, accounting, and billing. A metering process called exporter located at a router or switch analyzes network traffic and

aggregates information about the network traffic. The exporter then transmits the flow of information to the collectors. The data collected from the various exporters is subsequently used for network measurement.

The internal event detector needs to analyze the data packets in the network, and since the exporter in IPFIX collects data packets and has access to the data packets in the network, the internal event detector is built as an internal module in exporter in order to access the data packets and analyze them.

The exporter in IPFIX also sends the flow information to the collector. In addition, it can send user data. The exporter is used for building the internal event dispatcher, and the alert message is sent as part of the user data along with the flow information.

The collector is used for building the central event aggregator. This will aggregate the alert messages sent as part of the user-defined data by the exporters and pass them on to BotHunter and also send the signature to one or more internal event dispatchers.

### 4.7 Modifications to IPFIX-compliant Flow Generator

Libipfix [32] is a c-library that implements the IPFIX protocol was used for building the LTMS. The following changes were made in the exporter and collector.

- *Exporter*. This component was modified to implement the internal event detector. The internal event detector uses the *Boyer–Moore* [33] string search algorithm to analyze the traffic for signatures. A TCP socket server module was added to the exporter to receive signatures from the central event aggregator.

- *Collector*. This component was modified to implement the central event aggregator. It sends the signature received from BotHunter to one or many internal event dispatchers. Additionally, it sends the alert message received as part of an IPFIX message from the one or more exporters to BotHunter.
- *IPFIX Message*. A new flow template was created to include the alert message. The alert message consists of the time of detection of the event, which was necessary for BotHunter to correlate external and internal events, the type of event, and the signature id.

### 4.8 Modifications to BotHunter

BotHunter was modified to send the signature id along with the signature identified in the E3 or E4 event and the type of event, i.e., E3 or E4, to the central event aggregator. It was also programmed to receive alerts from the central event aggregator.

### 4.9 Detection Steps

- As shown in Figure 19, BotHunter detects an E3 through E4 event when the covert bot downloads the binary or command from the external peer.
- The signature used in indentifying the E3 through E4 event is propagated to the central event aggregator. The central event aggregator may be in the same computer as BotHunter or in a different computer, as shown in Figure 19.
- The central event aggregator propagates the signature to one or more internal event detectors.

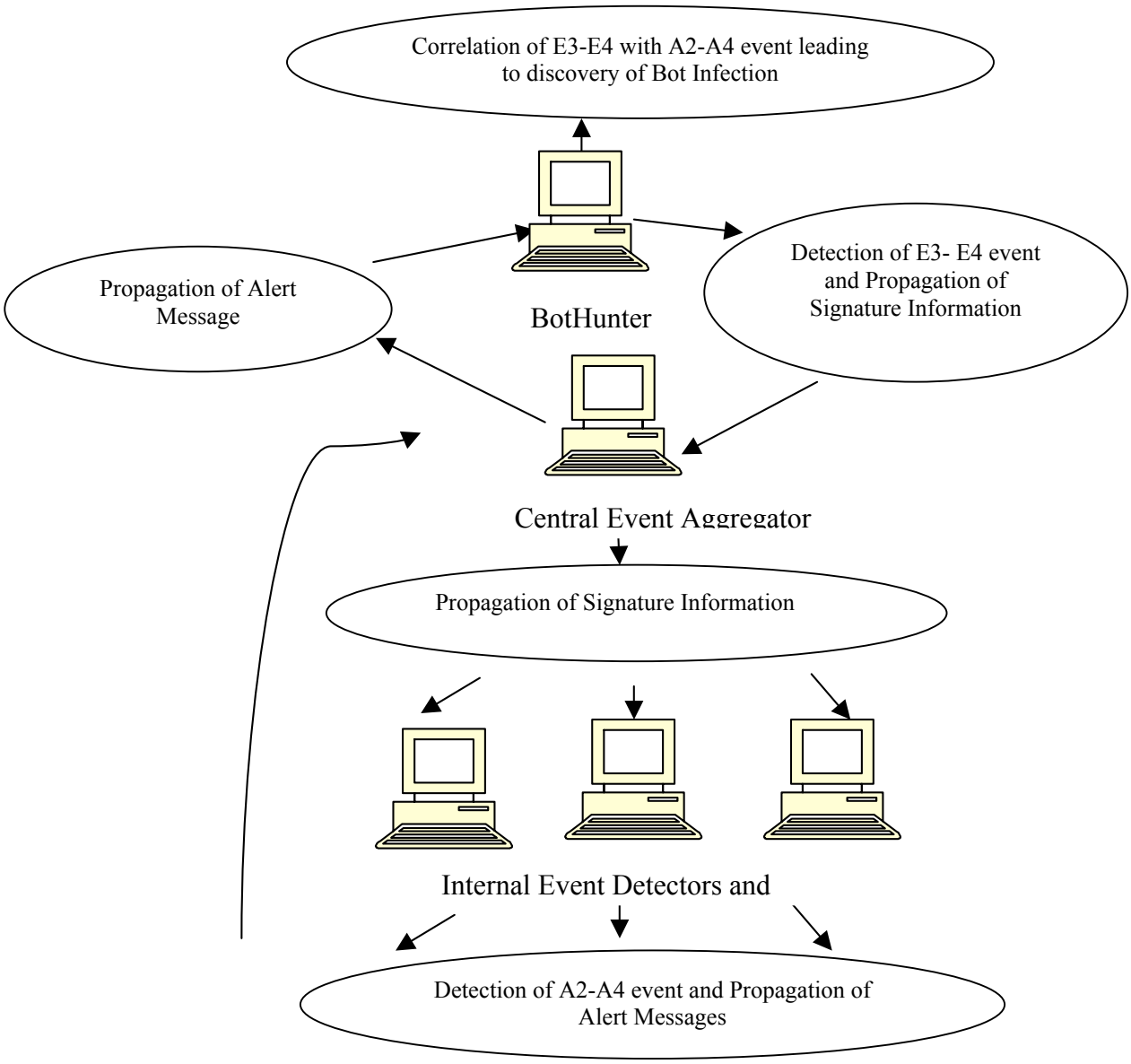


Figure 19. Detection steps.

- The internal event detector detects the A2-A4 events using the signature, and the internal event dispatcher propagates the alert message to the central event aggregator.
- The central event aggregator then forwards the alert message to BotHunter.
- If the E3 through E4 events and the A2 through A4 events are within the threshold time, a bot infection is declared.

## 4.10 Analysis of Defense Mechanism

### *4.10.1 Experiment Setup*

- *Covert Bot Application*. This application implemented the functionality of the covert bot. It was written in C++. It performs binary download, and command download from the remote command and control server. *External Peer Application*. This application simulated the remote command and control server. When a bot requests a bot binary, the bot binary it is sent through TCP.
- *Local Network*. As shown in Figure 20, the local network was simulated *on VMware consisting* of 40 computers. Thirty virtual computers running on the Ubuntu Linux Operating system were hosted on a VMware ESX server. These computers were connected to a virtual switch on the VMware ESX server. The other 10 computers running Ubuntu Linux Operating system were hosted on a VMware workstation that was running on top of a Linux server. The computers were connected to a physical switch.

- *Central Event Aggregator*. This application was run on a Linux desktop computer, as show in Figure 20.
- *Internal Event Detector and Dispatcher*. Two internal event detectors and dispatchers were used for the experiments. One of them was run on a virtual computer hosted on the VMware ESX server and monitored the traffic in the virtual switch, while the other was hosted on an Ubuntu Linux desktop and monitored the physical switch.

  I implemented and ran the Internal Event Detector, Dispatcher and the Centralized Event Aggregator on a desktop computer. These components utilized the signature database present in Snort. Using the single signature database enables them to analyze the network traffic with lesser CPU and memory utilization, which would be necessary if implemented on a network switch, since switches have limited memory and processing power.
- *External Network*. The external network consisted of a single Linux desktop computer that hosted the external peer application.
- *BotHunter*. This application was hosted on the same Linux desktop computer as the central event aggregator.

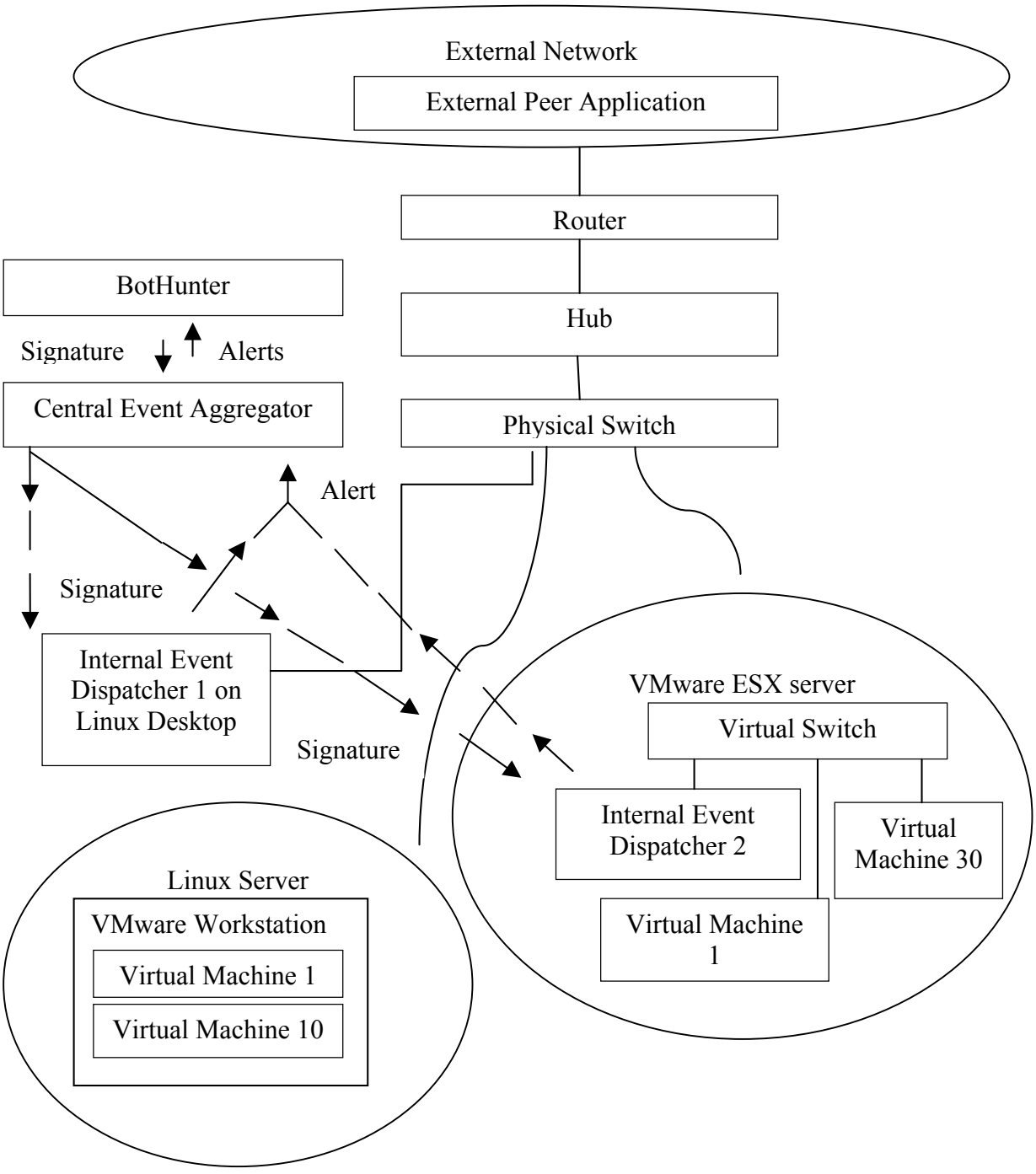


Figure 20. Defense experiment setup.

*4.10.2 CPU Utilization*

The CPU utilization of the internal signature detector application was dependent on the network bandwidth and on the signature database size. The internal signature detector analyzed the traffic for occurrences of signatures. If the signature database size was large and the volume of the traffic was high, the application spent more time in analyzing the traffic thereby increasing CPU utilization of the application.

This can be seen in Figure 21, wherein CPU utilization reached 100% when the network bandwidth was more than 15000 KBs per second and the signature database size was 350. However, if the size of signature database was 5, CPU utilization did not go beyond 45% even with a network bandwidth of 21000 KBs per second.

The internal signature detector had to be present at vantage points in the local network to analyze the traffic and detect the A2 through A4 events. Having a huge database of signatures at each and every location would not only increase redundancy but also increase CPU utilization when analyzing traffic. Hence, it made it advantageous to have only one centralized database of signatures available in BotHunter and propagate the signature detected during an E3 through E4 event to one or more internal signature detectors, which in turn use them to analyze the local traffic and detect the A2 through A4 events.

*4.10.2 Network Data Processed versus Signature Database Size*

This experiment was conducted with 4140.4 KB (size of the covert bot binary) of data being transferred in the network. As the network data was being transferred, the

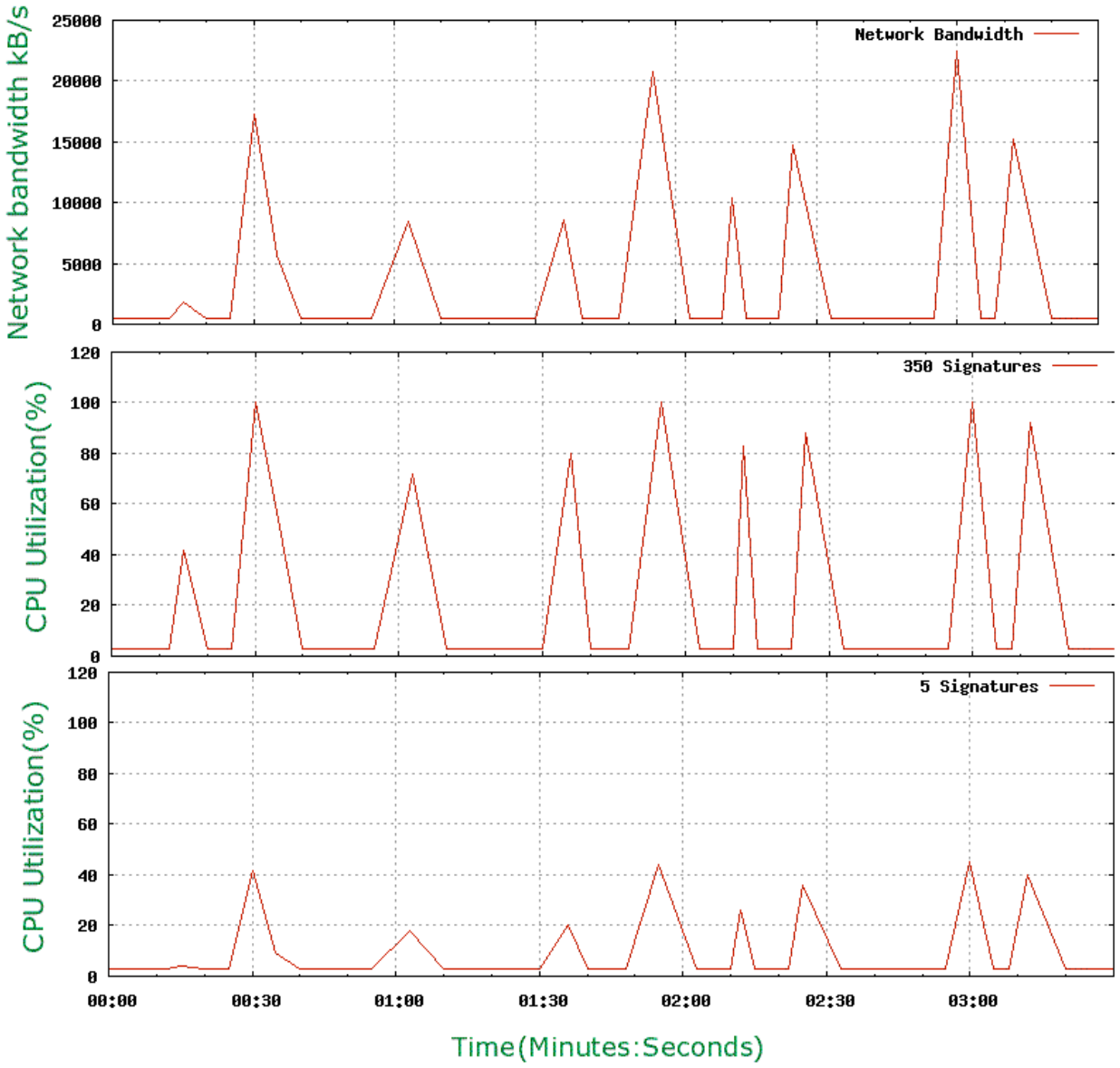


Figure 21. CPU utilization.

internal signature detector copied it in to its internal buffer to compare the received data with the signatures in the database. As the signature database size increased, the amount of network data analyzed by the internal signature detector decreased, since the internal buffer could not be analyzed quickly because of the increased signature database.

As illustrated in, Figure 22 the internal signature detector was able to analyze all the traffic in network when the database size was less than 6; however, when the database size increased, the size of data being analyzed by it decreased. This could potentially lead to missing the internal binary or commands being propagated in the network. The experiment was conducted three times.

*4.10.4 Time Delay*

The LTMS had to receive the signature identified by BotHunter without much time delay. As illustrated in Figure 23, when there was a time delay of 0.3 seconds, some of the 30 bots participating in the A3 events were not detected. This is because the covert bot propagated the binary (A3) to the rest of the botnet before the signature was received by the LTMS. A further increase in the time delay decreased the number of bots detected. The experiment was conducted three times. When the time delay was 0.8 seconds the average number of bots detected was 22 bots. Thus, it is critical to transfer the signature identified by BotHunter to the LTMS as soon as possible.

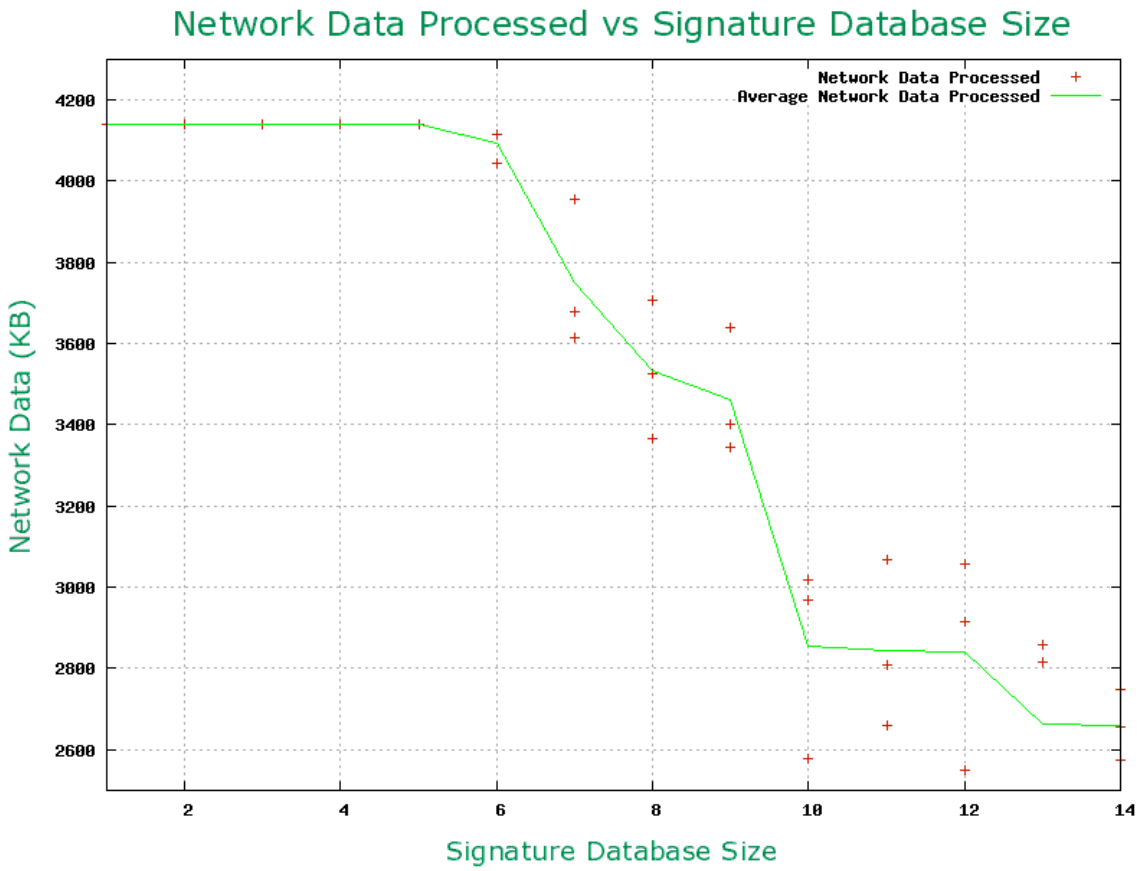


Figure 22. Network data processing.

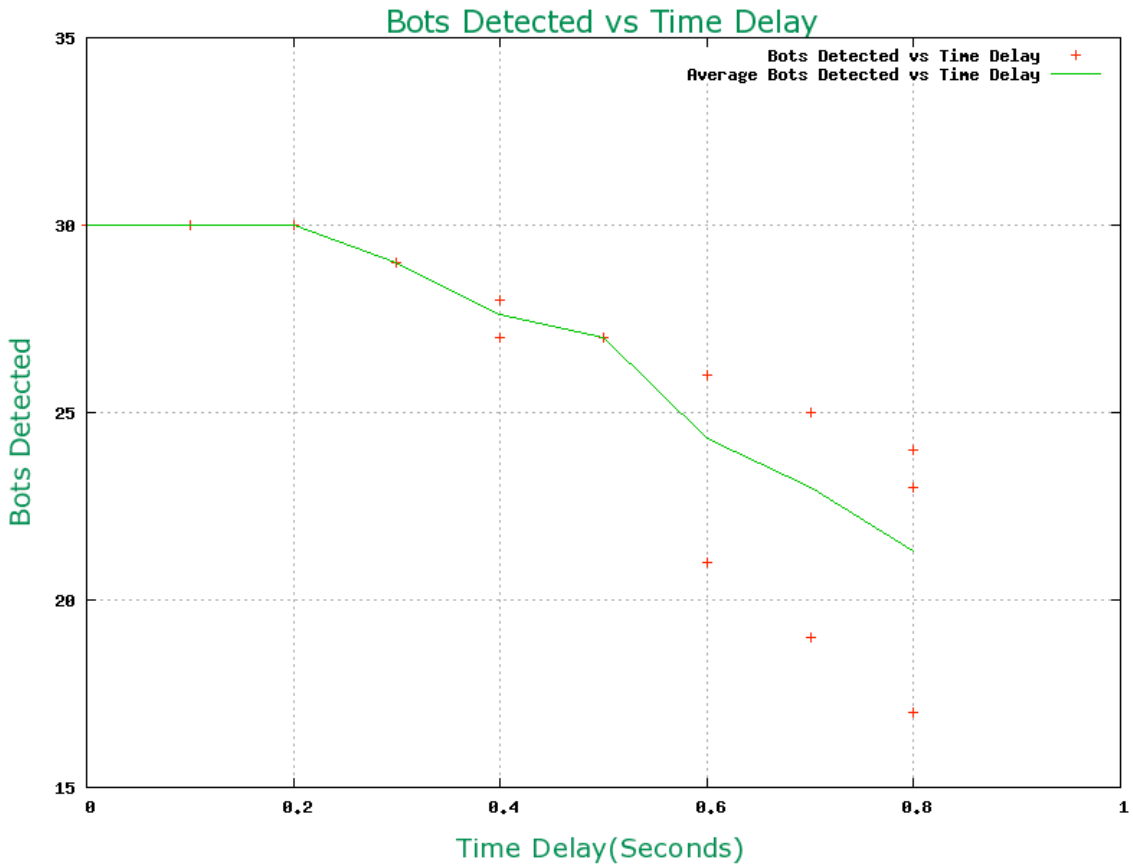


Figure 23. Bots detected vs time delay.

# CHAPTER 5

# CONCLUSION

## 5.1 Contribution

Security analysts create new defense mechanisms against attacks, and attackers find new ways of defeating existing defense mechanisms to create havoc for private computer users and businesses alike. This cat and mouse game will continue in the future. Technologies that benefit the people are hijacked by attackers for malevolent purposes. IRC channels have been developed for people to communicate, exchange files, etc. However, this technology has been misused by attackers to create botnets to carry out malicious activities. Initially, botnets created by attackers were based on a centralized command and control, and techniques were devised accordingly to detect these botnets.

However, attackers have come up with botnets that utilize the peer-to-peer networks which are harder to detect. Hence, it is necessary for researchers to detect vulnerabilities in current defense mechanisms before the attackers can find them. The authors of [12] have proposed a new model of bot infection that is different from existing models. It is not only necessary to propose a new model but also demonstrate that the new model is viable by implementing and fine-tuning it by conducting experiments.

This thesis presents implementation of the framework for a covert botnet communication in a private subnet. The purpose of said implementation is to show that the new model of bot infection is as potent as the existing botnet models while doing a better job of maintaining stealth and evading detection from current detection mechanisms such as BotHunter. I carried out experiments by simulating an infection in a computer network using the new model and showed that the attack is successfully able to

maintain stealth and evade detection. I have also suggested some improvements to the new model of bot infection to make it more robust.

Research in computer security should not stop at finding vulnerabilities. Rather, it should leave no stone unturned to finding ways to prevent detection and exploitation of vulnerabilities by attackers. Consequently, in this thesis, I have also designed a defense mechanism against the new model of bot infection.

This design involves monitoring the local traffic within a network using the signature identified by BotHunter during an external event (E3, E4) and sending alert messages using the IPFIX protocol to BotHunter when an internal event (A2-A4) is detected. This enables BotHunter to correlate external events with internal events to detect bot infection. I have implemented the new design and carried out experiments, wherein I simulated a bot infection in a computer network and successfully detected the bot infection.

### 5.2 Future Work

There is no defense mechanism that is absolutely foolproof, so there is scope for improving the defense mechanism proposed and implemented in this paper. Firstly, the implementation used in this paper uses only substring matching for signature identification which is sufficient for an A2, A3, or A4 internal events. This could be improved by adding more complex rule matching techniques like regular expressions.

Secondly, when there is a hierarchical structure of switches in a network, redundancy in the alert messages being sent to BotHunter occurs. For instance, if a bot in a computer connected to switch 1 propagates a binary to another bot hosted in another computer connected to switch 2, the internal signature detectors monitoring switch 1 and

switch 2 both send alert messages to the central event aggregator. Techniques to eliminate this redundant information being propagated could be another source of future work.

Thirdly, the defense mechanism works fine when the computer that performs the Binary, Command, Peer List download (E3, E4 external events), resides in the same subnet while performing the internal propagation (A2-A4 internal events). This allows my defense mechanism to detect both the external event and the internal events. This may not be true in all scenarios as illustrated by the following case. People carry their laptops from home to the workplace or college and the subnets they are part of changes as they do it. A bot in a Laptop may perform the E3, E4 external events when the host is connected to the home subnet and then when the victim carries the laptop to the workplace the bot subsequently perform the binary propagation to the other computers in the workplace.

The Local Traffic Monitoring System would not be able to detect the internal propagation as the BotHunter in the workplace is unaware of the external events. Detection of such infections would be another improvement to the existing Defense mechanism.

Finally, the concept of using a centralized signature database and using the signature detected at the router and propagating to monitors present within network switches can be extended to other devices such as PDA's, Mobile Phones, etc., which like the Network Switch have less memory and processing capabilities and replicating the signature database in them will be redundant. In the future Ubiquitous computing will become popular and households will contain ambient devices with networking

capabilities. These devices can be potentially be exploited by attackers. Hence Monitors can be built into the ambient devices which utilize a centralized database of signatures, this would reduce the processing and memory needed to analyze the network traffic.

# REFERENCES


1. Cooke, E., Jahanian, F., and McPherson, D. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet* Workshop, 2005.

2. Ianelli, N. and Hackworth, A. *Botnets as a Vehicle for Online Crime*. Technical Report, CERT Coordination Center, 2005.

3. Strayer, W.T., Walsh, R., Livadas, C., and Lapsley, D. Detecting botnets with tight command and control. *In Proceedings of 31st IEEE Conference on Local Computing Networks*, 2006.

4. Federal Bureau of Investigation. *News Releases*. 2008. http://www.ic3.gov/media/ initiatives/fbibotroast.pdf. September 2008.

5. Stewart, J. *Storm Worm DDoS Attack*. SecureWorks. 2008. http://www.secureworks.com/ research/threats/storm-worm. March 2008.

6. Porras, P., Saidi, H., and Yegneswaran, V. *A Multi-perspective Analysis of the Storm (peacomm) Worm*. Technical Report, Computer Science Laboratory, SRI International, October 2007.

7. Wikipedia. *Storm Botnet*. 2008. http://en.wikipedia.org/wiki/Storm_botnet. March 2008.

8. Wikipedia. *Botnet*. 2008. http://en.wikipedia.org/wiki/Botnet. February 2008.

9. F-Secure. 2008. http://www.f-secure.com/weblog/archives/00001580.html. March 2008.

10. Daswani, N. and Stoppelman, M. The anatomy of Clickbot.A In *Proceedings of 1st Conference on First Workshop on Hot Topics in Understanding Botnets*, 2007.

11. Federal Bureau of Investigation. News releases. 2008. http://losangeles.fbi.gov/ dojpressrel/pressrel08/la041608usa.htm. September 2008.

12. Shirley, B. and Mano, C.D. A model for covert botnet communication in a private subnet. In *Proceedings of IFIP Networking*, 2008.

13. Gu, G., Porras, P., Yegneswaran, V., Fong, M., and Lee, W. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium* 2007.

14. Canavan, J. The evolution of malicious irc bots. In *Proceedings of Virus Bulletin Conference*, 2005.

15. Saha, B. and Gairola, A. *Botnet: An Overview*. White Paper, CERT-In, June 2005.

16. M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. *In Proceedings of the 6th Internet Measurement Conference*, 2006.

17. Grizzard, J.B., Sharma, V., Nunnery, C., ByungHoon Kang, B.B-H., and Dagon, D. Peer-to-peer botnets: Overview and case study. In *Proceedings of 1st Conference on First Workshop on Hot Topics in Understanding Botnets*, 2007.

18. Freiling, F., Holz, T.,and Wicherski, G. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of- service attacks. *In Proceedings of 10th European Symposium on Research in Computer Security*, 2005.

19. Guofei, G., Zhang, J., and Lee, W. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of 15th Annual Network and Distributed System Security Symposium*, 2008.

20. Baecher, P., Koetter, M., Holz,T., Freiling, F., and Dornseif, M. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of 9th International Symposium on Recent Advances in Intrusion Detection*, 2006.

21. Scarlata, V., Levine, B.N., and Shields, C. Responder anonymity and anonymous peer-to-Peer file sharing. In *Proceedings of the IEEE International Conference on Network Protocols*, 2001.

22. Vogt, R. and Aycock, J. *Attack of the 50-Foot Botnet*. Technical Report, 2006-840-33, Department of Computer Science, University of Calgary, August 2006.

23. Landesman, M. Antivirus software. About.com. 2008. http://antivirus.about.com/od/virusdescriptions/p/nugache.htm. September 2008.

24. Holz, T., Steiner, M., Dahl, F., Biersack, E., and Freiling, F. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats,* 2008.

25. Snort.org. *Snort Open Source Network Intrusion Prevention and Detection System*. 2009. http://www.snort.org/. June 2008.

26. Roesch, M. Snort - lightweight intrusion detection for networks. In *Proceedings of USENIX LISA*, 1999.

27. Lee, Y.S., Shin, S.H., and Kwon, T.G. Signature-aware traffic monitoring with IPFIX. In *Management of Convergence Networks and Services*, Springer, 2006, 82-91.

28. Claise, B., Bryant, S., Leinen, S., Dietz, T., and Trammell, B.H. IPFIX protocol specifications. Internet draft, draft-ietfip_x-protocol-26, Sep. 2007.

29. Wikipedia. IPFIX. 2008. http://en.wikipedia.org/wiki/IP_Flow_Information_ Export. September 2008.

30. Portable Thread Library. *GNU Pth*. 2007. http://www.gnu.org/software/pth/. March 2008.

31. VMware. *Virtualization Software*. 2008. http://www.vmware.com/. May 2008.

32. IPFIX Library. 2008. http://www.fokus.fraunhofer.de/en/net/more_about/download/ipfixlib.html. September 2008.

33. Robert Stephen Boyer, R.S., and Moore, J.S. A fast string searching algorithm. *Commun. ACM 20*, 10 (1977), 762-772.