

BOTNET DETECTION AND MITIGATION USING HONEYPOTS

A Dissertation submitted in partial fulfillment of the requirement for the

Award of degree of

MASTER OF TECHNOLOGY

IN

INFORMATION SYSTEMS

Submitted By

LIPIMONI TAYE

(2K13/ISY/11)

Under the esteemed guidance of

Dr. N.S. RAGHAVA

Associate Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

Session 2013-2015

CERTIFICATE

This is to certify that the thesis entitled “**Botnet Detection and Mitigation Using Honeypots**” submitted by Lipimoni Taye, Roll No. 2K13/ISY/11 student of Master of Technology (M. Tech.) in Information Systems from Department of Computer Science and Engineering, Delhi Technological University, Delhi is a bonafide record of the candidate’s own work carried out by her under my guidance.

Signature

Dr. N.S. Raghava

Associate Professor

Department of Electronics & Communication

Delhi Technological University

Place: DTU, Delhi

Date: _____

ACKNOWLEDGMENT

I would like to express my deep sense of gratitude towards my supervisor **Dr. N. S. Raghava**, *Associate Professor, Department of Electronics and Communication* for his able guidance, support and motivation throughout the time. It would not have been possible without the kind support and help of many individuals and **Delhi Technological University**. I would like to extend my sincere thanks to all of them.

I would like to express my gratitude and thanks to **Dr. O.P Verma (Head of Dept.)** for giving me such an opportunity to work on the project.

I would like to express my gratitude towards my **parents & staff** of Delhi Technological University for their kind co-operation and encouragement which helped me in completion of this project.

My thanks and appreciations also go to my **friends and colleagues** in developing the project and people who have willingly helped me out with their abilities.

Lipimoni Taye

Roll No: 2K13/ISY/11

M. Tech. (Information Systems)

Delhi Technological University

ABSTRACT

The technological advances in the fields of networking and distributed computing have been in an increasing growth trend. And as the curve rise, the ease of access to these remote computers and networking devices for carrying out various measures like communication, data transmission and various transactions have also seen a significant increase especially in the recent years. Security of such infrastructure have always been a major concern for both hardware and software vendors. Although a number of solutions have been proposed and implemented over the time to combat the security vulnerabilities, the impact of the black hats (attackers and hackers) are always at par with those of the security defenders.

Botnet phenomenon is one such threat to the security of any network systems and its end users which has already gained a widespread negative impact worldwide. Threats via a botnet include a wide range of malicious activities like spam distribution, malware, phishing, launching DDoS/DoS, identity theft, illegal resource utilization and many more. Therefore it has become an utmost necessity from security point of view to come up with an efficient and robust botnet detection and mitigation technique. While a wide range of solutions have already come up in the research history, the honeypot technology in detection of bots and botnets is one significant approach and is still in its infant stage.

This thesis aims towards having a complete understanding of the botnet phenomenon, its architecture, types, lifecycle and various detection mechanisms. The honeypot technology is also studied in depth especially in context to its application in network security. Deployment of honeypots for detecting bots and botnets being the primary goal of this thesis, work is done to implement a centralized botnet command and control architecture. A low interaction honeypot is deployed in the compromised bot machines and servers to perform a monitoring activity for the attackers trying to connect and also logging information for further study and analysis.

Table of Contents

Chapter 1 INTRODUCTION.....	1
1.1 The Botnet Threat	3
1.1.1 Botnet Phenomenon	3
1.1.2 History.....	5
1.2 Botnet Architecture	7
1.2.1 Centralized Botnets	8
1.2.2 Decentralized (P2P) Botnet:	9
1.3 Botnet Detection.....	10
1.4 LifeCycle of a Botnet.....	13
1.5 Related Work	14
1.6 Organization of the Thesis	16
Chapter 2 HONEYPOTS AND THEIR TYPES	18
2.1 Introduction to Honeypots	18
2.2 Types of Honeypots	18
2.2.1 Research honeypots	18
2.2.2 Production honeypots.....	19
2.2.2.1 Prevention	19
2.2.2.2 Detection	19
2.2.2.3 Response	20
2.2.3 Honeynets	20
2.3 Level of Interaction.....	21
2.3.1 Low-interaction Honeypots	21
2.3.2 Medium-interaction Honeypots	22
2.3.3 High-interaction Honeypots	22
2.4 Honeypots in Network Security	22
2.5 Honeypots for detection of bots and botnets.....	23
Chapter 3 PROPOSED METHODOLOGY	25
3.1 Scope of the Methodology and Main Goals	25
3.2 Implementation idea.....	25
3.2.1 Command and Control Server Implementation	25
3.2.2 Bots finding the C&C	27
3.2.3 The Bot model and transitions	27

3.2.4 Bot reporting C&C.....	28
3.2.5 Tracking and Detecting Bots	29
Chapter 4 EXPERIMENTAL SETUP AND IMPLEMENTATION	30
4.1 Botnet Implementation.....	30
4.2 Communication Methodology	32
4.3 Detection with Honeygot	35
4.4 Deployment of the Honeygot.....	36
4.5 Key Implementation Features of the Honeygot used	37
4.6 Honeygot Graphical User Interface	38
4.7 Internal Mechanism of Honeygot Application	39
4.7.1 Launching and Initializing of the Honeygot	40
4.7.2 Initialization of the LIModule.....	41
4.7.3 LIProtocol (FTP protocol) interaction with Client	43
Chapter 5 EXPERIMENTAL OBSERVATION AND DISCUSSION.....	46
Chapter 6 CONCLUSION AND FUTURE WORK.....	56
6.1 Conclusion	56
6.2 Future Scope	58
Chapter 7 REFERENCES.....	60

LIST OF FIGURES

Figure 1.1 Design Overview	2
Figure 1.2: Taxonomy of Botnet Architecture.....	7
Figure 1.3 Centralized Botnet Architecture	8
Figure 1.4 Decentralized P2P Botnet.....	9
Figure 1.5 Botnet Detection Techniques	11
Figure 1.6 Botnet LifeCycle	13
Figure 2.1 Honeynet Setup	21
Figure 3.1 Design of Botnet C&C	26
Figure 3.2 Various Transitions of bots	28
Figure 4.1 Botnet Implementation Flowchart.....	31
Figure 4.2 Multithreaded design showing two clients connected at the same port.	38
Figure 4.3 Honeypot GUI	39
Figure 4.4 Honeypot Application Launch Flow	41
Figure 4.5 Flow of Events in LIModule	42
Figure 4.6 FTP communication between the client (Botmaster) and server (Honeypot)	43
Figure 5.1 Client side console output (client-server in same network)	47
Figure 5.2 Client side console output(client-server in different network).....	47
Figure 5.3 Server side console output.....	49
Figure 5.4 Bots Table at botmaster database	49
Figure 5.5 Graphical User Interface of the low interaction honeypot	51
Figure 5.6 Honeypot GUI with client connected	52
Figure 5.7 Botmaster telnet window when bot is connected	53
Figure 5.8 Console output of the log file created by honeypot.....	54
Figure 5.9 Log file saved in text format	55

LIST OF TABLES

Table 1.1 Historical list of Botnets(Timeline)	6
Table 2.1 Classification of Botnet Commands	24
Table 4.1 Main Classes of the botnet Package	34
Table 4.2 Bots Table	35

Chapter 1 INTRODUCTION

One of the most serious security risks to the internet and its vast users in today's world is Botnet. A botnet is a network of compromised computers that are remotely controlled by a Botmaster or Bot herder under a Command and Control (C&C) architecture. Botnets today is widespread, reason pertaining not solely on malice but profits instead. Delay in detection of new and emerging botnets leads to higher profits for the adversaries. Bots under a botnet are responsible for sending spams, malicious packets, performing Distributed Denial of service attacks, phishing, click frauds and many more. The detection of botnet has become an important topic of research over the years and many approaches to detect and mitigate botnets were studied and implemented. The centralized botnet architecture provides a simple and real-time communication platform to the bot controllers. Detection of centralized botnet aids in tracking down the botmaster and also helps to mitigate some of the destructive capabilities of a botnet.

Several botnet detection techniques have been studied and out of them Honeypot technique has found importance among researchers. A honeypot is a decoy system which is deployed in an existing network to detect any unauthorized use of resources. Honeypots helps to understand and analyse botnet phenomenon and its characteristics. Honeypots can be classified into low interaction, high interaction and hybrid honeypots. Low interaction honeypots are used to emulate a limited number of services, high interaction honeypots either emulate a complete operating system and sometimes even use real installations of OS. The classification is basically dependent on the level of interaction with the attacker. A network of two or more honeypots is termed as a honeynet. With the help of honeynets, the botmasters or bot controllers can be tracked down.

In this thesis, a detailed study on botnet phenomenon, its architecture, mechanism and detection techniques has been done. The aim is to implement centralized botnet architecture over standard FTP, IRC and HTTP protocols and detecting it using the honeypot detection technique. The honeypot logs are used to study the source and behaviour of any malicious attacks. Low interaction honeypots are comparatively easy to deploy and maintain. It also entails lower risks than the high interaction ones with better scalability and overall less

resource intensive. Also most of the botnets use centralized command and control (C&C) architecture to issue commands and control the bots. The P2P variant of botnet is also becoming quite popular but the centralized infrastructure provides a central point of failure and is also useful in terms of security aspects.

The broad design overview of this particular thesis work is depicted in the following Figure 1.1

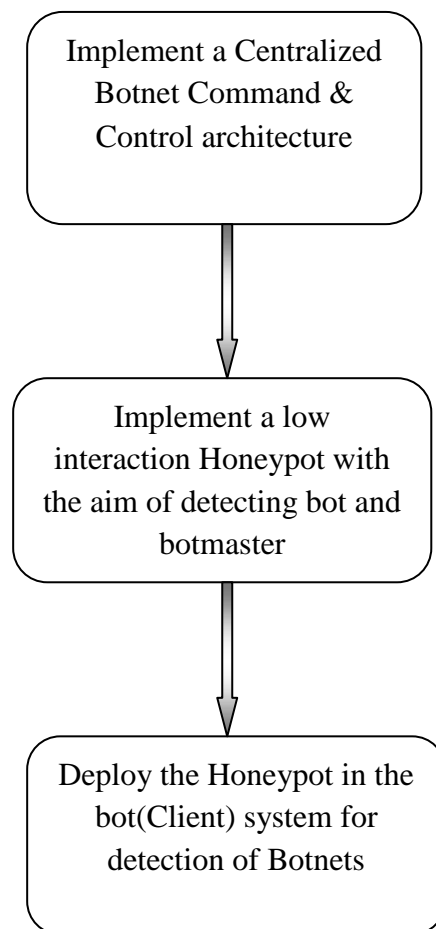


Figure 1.1 Design Overview

1.1 The Botnet Threat

A botnet is comprised of a network of bots or zombies which are compromised by malware, spams or Trojan without the knowledge of the user. The term bot as derived from Robot is programmed to perform some automated functions remotely controlled by a botmaster or a botherder via a Command and Control(C&C) architecture. Lured by the financial gains in recent years, botnet related attacks have increased significantly over the years. Botnets provide a distributed platform for cyber criminals to launch distributed denial of service attacks, send Trojan, spams, and phishing emails, media piracy, theft of useful computing resources and information, extortion of e-commerce business, perform click frauds, adware, spyware, fast flux etc. In order to understand the botnet phenomenon, systematic explanation of different botnet features is needed [1]. Bot distribution is widely spread in the internet [2] when vulnerable and unprotected computers are left in open for the attackers. They are infected and reports are sent to the botmaster. The bots stay inactive until they receive commands from the botmaster to perform any malicious operations.

1.1.1 Botnet Phenomenon

The major activities in a botnet phenomenon can be classified into three parts: (1) *Searching* for unprotected or vulnerable hosts in the internet, (2) *Distribution* of the bot code to the target computers to form a network of bots and (3) *Signing On* or connection of the bots to the botmaster to start performing actions through the command and control mechanism. Botmasters are particular about building a robust C&C architecture to manage millions of globally distributed bots. They had a good understanding of networking and protocols. IRC(Internet Relay Chat) channels were used as the C&C centres for the first of its kind botnets. The use of IRC initiated the trend towards centralized command and control. One of the most notorious and earliest bots names SDBot or Spybot was programmed in C++ which later began to exploit the vulnerabilities of the Microsoft windows platform. RBot (2003) started using compression and encryption schemes/ packers such as UPX, Morphine, and ASPack. With these new demands, a requirement of skilled coders with a clear understanding of encryption schemes, cryptography emerged. However the centralized C&C infrastructure of botnet as shown in figure 1.2 is vulnerable for detection and single point of failure. Hence the emergence of new generation botnets with Peer to Peer (P2P) architecture without a

centralized server [2] for example the Sinit(2003) and Phatbot(2004). Storm Worm/Nuwar(2007) botnet with decentralized P2P architecture was highly complex and difficult to combat. The botnet threat spread like a global pandemic. As per McAfee labs detection botnet infection is more than 1 million in many countries which include India. According to recent Kaspersky lab statistics, in Q1 of 2015, cybercriminals committed more than 23000 botnet-assisted DDoS attacks on web resources residing in 76 different countries. The servers of USA, China and Canada are on the frequent list of targets while resources of Europe and Asia Pacific region also under serious threat.

The botnet technology is driven by challenges from the many security solutions existing in the market. The advancement of bots and botnets also results in driving security technologies to bring forward new and complex security measures and countermeasures to combat challenges by new and emerging botnets. Driven by monetary benefits the challenges and risks in botnet technology are in an increasing trend making the tasks of security professionals even more critical. Botnet detection and mitigation have been a major research agenda and over the years and many defensive measures were also proposed. But the arms race between the criminals and defenders is still on-going. Current botnets use state of the art techniques to challenge the existing detection methods. Using techniques like polymorphism and metamorphism, mutation of bot codes is possible without any disruptions to the semantics of the payload. Different bot binaries may exist in a single botnet. The traditional signature based detection methods looks for particular pattern of data in the binaries. Hence these methods fail to identify all obfuscated bots.

Recent botnets have also seen evolution of the traditional Command and Control architecture. Earlier a botmaster used to control the bots via a common control channel by utilizing centralized C&C mechanisms like IRC, HTTP protocols etc. In case of IRC, the botmaster takes benefit from an IRC server in a public IRC network by specifying a channel which is kept open for bots to log in to chat rooms. The bots used to scan a network and attack machines with vulnerabilities. Once the machine was infected the bots would connect to a specific channel (chat room) and receive commands from the botmaster. IRC also have been used in taking screenshots from host machines, upgrade or download a bot. Few years ago, control of many botnets changed from IRC channels to websites using HTTP. The HTTP bots can install malicious software on remote computers which can be controlled from a remote website. The attackers send the malicious codes via spam or instant message with links to victims which when opened installs the exploit code in the victims machine without the

knowledge of the later. On successful exploitation, the victim machine can be remotely controlled for various malicious activities. Zeus bot(also called ZBot) was designed to steal bank credentials is a special type of HTTP bot. Both IRC and HTTP botnets are vulnerable to single point of failure. So in 2007 emergence of a new kind of botnet was seen which used P2P protocol. One of its kind name Win32/Nuwar(later known as Storm worm) used an encrypted implementation based on eDonkey protocol. It was majorly responsible for spam distribution in 2007-2008 till it was taken down. The flexible and distributed nature of the P2P botnet makes it more complex to combat. But maintaining such type of Botnet is also a major challenge.

Recently botnet attacks have invaded into mobiles devices [3]. As mobile devices are emerging with new technologies and features with a increased used of internet(eg 3G,4G and LTE technologies), with the increased use of such devices, the security of such devices are becoming a major concern in the cyber world. Nowadays smartphones provide some vulnerability to attract botmasters. Mobile botnet also works on three principle components: propagation, C&C and communication infrastructure. Mobile botnets use SMS/MMS or Bluetooth mechanism other than conventional IP network for communication. Looking into the rise in usage of internet in mobile devices, development of new IP based C&C mechanism (HTTP based) is seen. Researchers discovered botnet activities over wifi connections. Recent studies also observed the use of social networking as a platform to implement a Command and Control mechanism for mobile botnets.

1.1.2 History

The takeover of botnets for malware attacks and intrusion purposes started during the early nineties. Early botnets used IRC channels as a medium until the emergence of C&C. In 2003 the hackers of Oregon in US launched a DDoS attack on eBay with the help of 20000 compromised bot hosts [4]. The emergence of P2P botnets was seen in 2004. The client of P2P itself was programmed and linked to servers taking Gnutella and WASTE to do the communication. In 2005, a newer and different kind of botnet virus named Zotob started DDoS attacks to various websites in the United States. In early 2006, statistics derived from CBI and Microsoft declared that around 57,783 hosts were infected by botnet and it rose to around 88,136 by September which was indeed an threatening increase. In April 2008 , the infection from the world's largest botnet till then named Kraken botnet which compromised at least 50 of the fortune 500 companies and the host capture count was over 400000 bots.

Damballa released the kraken botnet recovering around 495000 bots [5]. Zeus botnet was also a very notorious botnet, emerged in 2012 was intended to rob about \$47 million banking customers from Europe. The historical list of botnets with maximum estimated bots is shown in the following table [6].

Year	Name	Number of estimated bots	Spam capacity (billion/d)	Aliases	Detection approach	Type	Reference
2008	Conficker	10 500 000+	10	DownAndUp, Kido	AV software	HTTP/P2P	Schmudlach (2009)
	Mariposa	12 000 000	-	-	Manual	IRC/HTTP	McMillan (2010)
	Sality	1 000 000	-	Sector, Kuku, Kookoo	Manual	P2P	Falliere (2011)
	Asprox	15 000	-	Danmec, Hydraflux	Symantec	HTTP	Goodin (2008)
	Gumblar	n/a	-	-	Manual	HTTP	Mills (2009)
	Waledac	80 000	1.5	Waled, Waledpak	Kaspersky	SMTP/P2P	Goodin (2010)
	Onewordsub	40 000	1.8	N/A	-	SMTP	Keizer (2008)
	Xarvester	10 000	0.15	Rlsloup, Pixoliz	McAfee	SMTP	Symantic (2010)
	Mega-D	509 000	10	Ozdok	Manual	HTTP	Warner (2010)
	Torpig	180 000	-	Sinowal, Anserin	ESET	HTTP/IRC	Miller (2009)
	Bobax	185 000	9	Bobic, Oderoor, Cotmonger	Manual/ BitDefender	HTTP	Symantic (2010)
	Lethic	260 000	2	None	Symantec	IRC	Symantic (2010)
	Kraken	495 000	9	Kracken	Scan IP addresses	IRC	Jackson (2008)
2009	Maazben	50 000	0.5	-	-	SMTP	Symantic (2010)
	Grum	560 000	39.9	Tedroo	FireEye researchers	SMTP	Danchev (2009)
2008	Conficker	10 500 000+	10	DownAndUp, Kido	AV software	HTTP/P2P	Schmudlach (2009)
	Mariposa	12 000 000	-	-	Manual	IRC/HTTP	McMillan (2010)
	Sality	1 000 000	-	Sector, Kuku, Kookoo	Manual	P2P	Falliere (2011)
	Asprox	15 000	-	Danmec, Hydraflux	Symantec	HTTP	Goodin (2008)
	Gumblar	n/a	-	-	Manual	HTTP	Mills (2009)
	Waledac	80 000	1.5	Waled, Waledpak	Kaspersky	SMTP/P2P	Goodin (2010)
	Onewordsub	40 000	1.8	N/A	-	SMTP	Keizer (2008)
	Xarvester	10 000	0.15	Rlsloup, Pixoliz	McAfee	SMTP	Symantic (2010)
	Mega-D	509 000	10	Ozdok	Manual	HTTP	Warner (2010)
	Torpig	180 000	-	Sinowal, Anserin	ESET	HTTP/IRC	Miller (2009)
	Bobax	185 000	9	Bobic, Oderoor, Cotmonger	Manual/ BitDefender	HTTP	Symantic (2010)
	Lethic	260 000	2	None	Symantec	IRC	Symantic (2010)
	Kraken	495 000	9	Kracken	Scan IP addresses	IRC	Jackson (2008)
2009	Maazben	50 000	0.5	-	-	SMTP	Symantic (2010)
	Grum	560 000	39.9	Tedroo	FireEye researchers	SMTP	Danchev (2009)
	Festi	n/a	2.25	Spamnost	ESET	SMTP/DoS	Morrison (2012)
	BredoLab	30 000 000	3.6	Oficla	Symantec	HTTP/SMTP	Crowfoot (2012)
	Donbot	125 000	0.8	Buzus, Bachsoy	Symantec	HTTP	Stewart (2009)

Table 1.1 Historical list of Botnets(Timeline)

1.2 Botnet Architecture

In order to combat the ever increasing botnet challenges, much research is focused towards new and complex methods of botnet detection and mitigation. This thesis work is also focused towards bringing forth a solution towards detection of current and future botnet related malicious attacks. The work mainly focuses on detection of Botnet based on its architecture and deployment of Honeypot based mitigation techniques.

Botnet characteristics are primarily based upon the type of architecture it is built in [7]. Based on this concept, the major architectures of botnet namely Centralized, Decentralized(P2P), Hybrid have been studied in detail and careful measures are taken to present a java based simulation for the most commonly used centralized botnet architecture.

The taxonomy of botnet architecture is shown in the following Figure 1.2

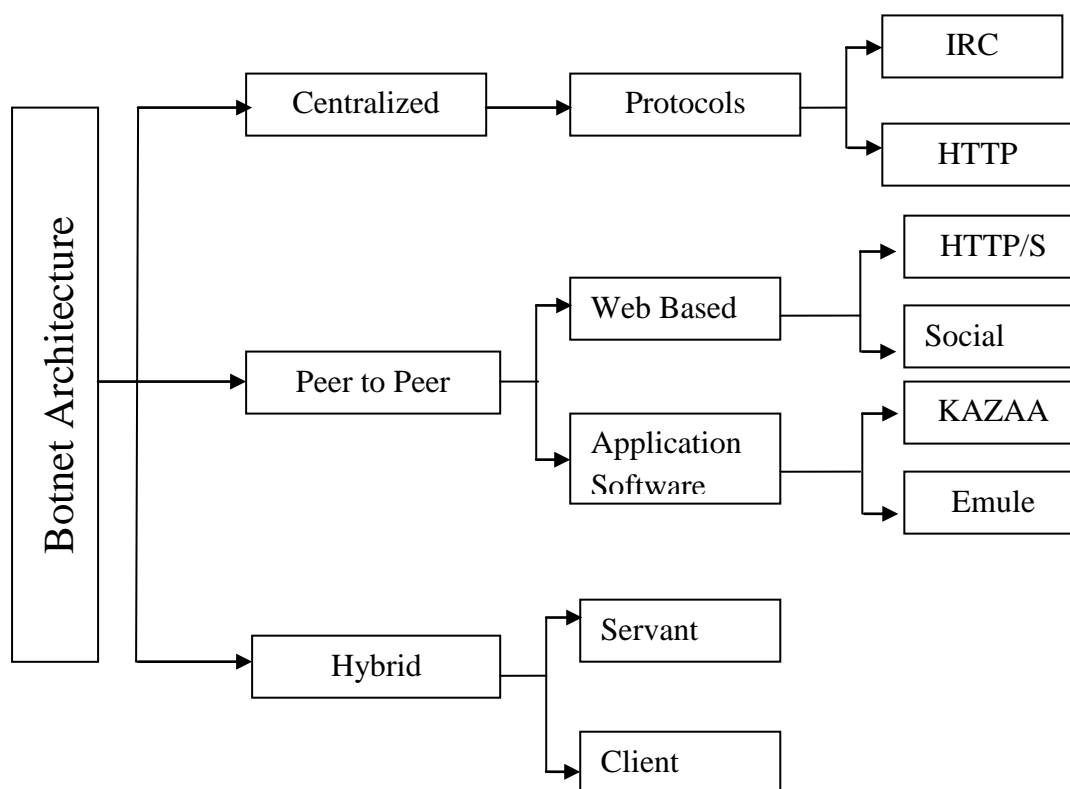


Figure 1.2: Taxonomy of Botnet Architecture

Based on its architecture botnets can mainly be classified into two types:

1.2.1 Centralized Botnets: The centralized C&C architecture is similar to the Client-Server architecture. Here the bots maintain a strong communication between one or more multiple connection points. Servers who send commands and control the bots are deployed on the connection points. IRC and HTTP are the main protocols in this type of architecture. Some advantages of centralized architecture are: 1) It is easy to deploy, 2) No requirement of any specific hardware, 3) No third party involvement as server is directly in contact with the bots due to which response is quick, 4) Direct communication between the bots and the botmaster gives better accessibility, 5) Updates in a timely manner and 6) Scalable. However the single point of failure feature of centralized architecture might make it vulnerable from an attackers point of view making it possible to bring down the entire botnet. Some of the popular IRC based C&C botnets are SpyBot, Agobot, SDBot, GT Bot etc. Figure 1.3 shows a Centralized C&C botnet architecture.

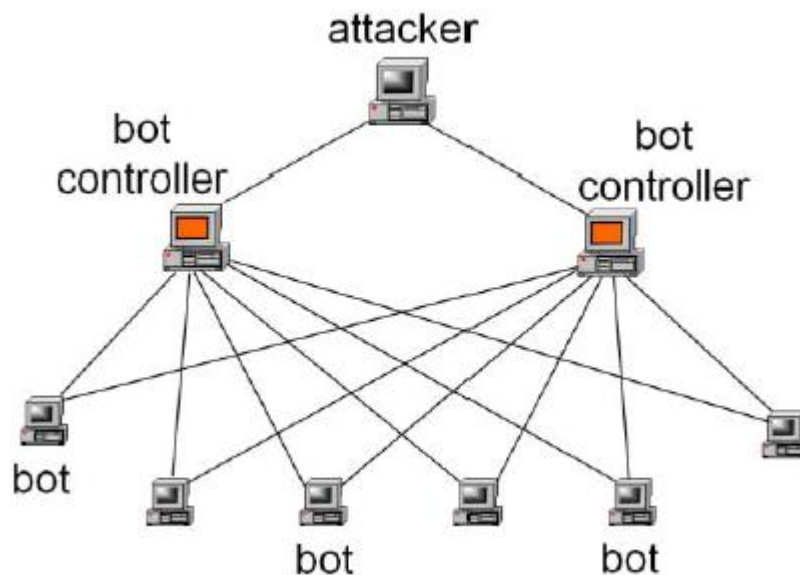


Figure 1.3 Centralized Botnet Architecture

1.2.2 Decentralized (P2P) Botnet: Unlike the centralized architecture a peer to peer botnet does not have a C&C server. Here the botmaster is in direct contact with individual bot peers which in turn communicates the command with other bots in the botnet. P2P botnet is more complex and difficult to suspend. However maintaining such botnets is not an easy task. The concept of single point of failure cannot be applied to P2P botnets which also make it difficult to diagnose the total area affected by the botnet. The interdependence between the bots in a P2P botnet is not strong. The newly arriving web based botnets are distributed and decentralized in nature. Social VPN is a P2P application which is free and allows computers to directly communicate in a shared community. Moreover it allows authenticated and encrypted communication. It provides Extensible Messaging and Presence Protocol(XMPP) supported backends like jabber.org and google chat etc and also a flawless access to remote files and servers. PhatBot, StormWorm etc are popular examples. Figure 1.4 shows a decentralized P2P botnet architecture.

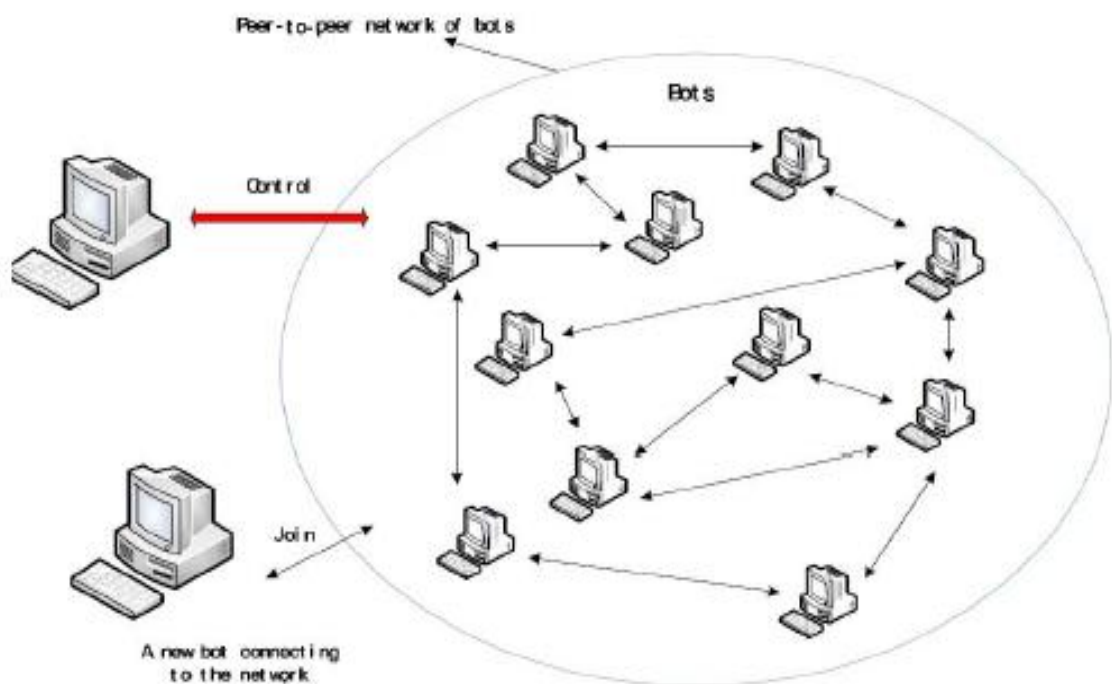


Figure 1.4 Decentralized P2P Botnet

[6]. Other than the two major classifications, botnets can also be of Hybrid architecture [8] which inherits the properties of both P2P and centralized architecture. This model follows the concept of Client bot and Servant bot. The servant bot behaves like a client and server and is configured with a routable IP address whereas the client bot is configured with a non-routable IP address and is not concerned with listening to connections. Servant bots listen for incoming connections through ports and update the bot peers with the IP address information.

Honey-pot Based detection techniques have come up to be an efficient mechanism in cyber security. A centralized architecture is chosen in this particular work for its various advantages and care has been taken to deploy the detector i.e. a low interaction honey-pot in a manner which allows targeting the botmaster and mitigating the various attacks from the adversaries.

1.3 Botnet Detection

With the eminent growth of botnet-related threats in the cyber world, detection and mitigation measures have become a very major concern. Botnet architecture and various C&C channels are also majorly responsible for the detection of botnets [9]. Over the years, different approaches have been studied and proposed in academia [10]. Out of them, deployment of a decoy system like honey-pots or honey-nets has been proposed and implemented for botnet tracking and monitoring. Studies from different papers reveal that honey-nets provide more insight towards understanding the botnet technology and its characteristics and also give an approach for detecting botnet-related infections. Honey-pots are computer systems with very less production value and are used as traps to seek the attention of cybercriminals to fall prey and hence track down the source of the attack. Honey-pots gather important information like [7].

- i. Bot signatures for content-based detection.
- ii. Information of botnet Command and control methodology.
- iii. Unknown security loop holes which enable bots to enter any network.
- iv. Tools and techniques used by the botnet controllers.
- v. The main motivation behind the attack.

Another approach of botnet detection is based on passive network traffic monitoring and analysis. This approach is beneficial to detect the existence of botnets. These techniques can broadly be classified into following four types. Figure 1.5 below shows the taxonomy of various botnet detection techniques followed by brief explanation of each type:

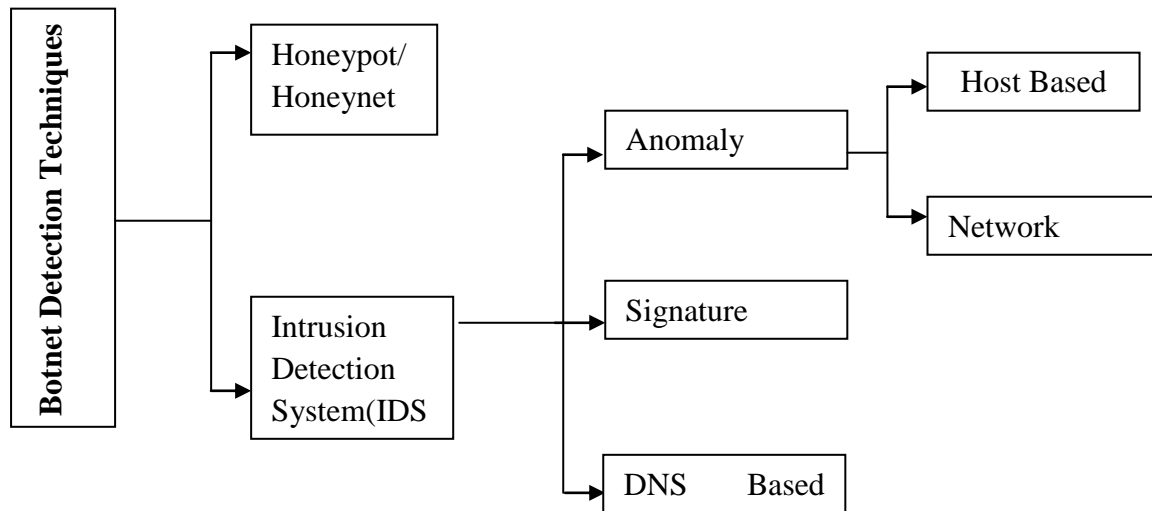


Figure 1.5 Botnet Detection Techniques

A. Signature Based Detection

This technique makes use of knowledge of important signatures and existing botnet behaviors. The benefits of this technique are immediate detection and very less false positives. Snort is an example of Signature based detection which is an open source Intrusion Detection System that monitors network traffic to detect intrusion. It is configured with a set of rules to find traffic which are suspicious. This method however fails to detect unknown botnets which means that zero day bots remain undetected when signature based detection approach is used.

B. Anomaly Based Detection

This kind of botnet detection technique performs detection on the basis of various network traffic anomalies such as huge volumes of traffic, high network latency, traffic directed towards unusual ports and peculiar system behaviour that could signify presence of malicious bots [11]. This technique can detect unknown botnets but sometimes it might detect an IRC botnet which has not yet been used for attacks. Several algorithms were proposed by

researches [12], one involved combination of TCP based anomaly detection with IRC message statistics. One more algorithm that assisted in detecting encrypted botnet used anomaly detection with flow data in transport layer. Botsniffer proposed by Gu et al [13] used network based anomaly detection for identifying botnet command and control in local area network. False positive rate is low in such detection techniques.

C. DNS Based Detection

This type of detection is mainly based on specific DNS information generated by a botnet. It is quite similar to anomaly based detection techniques. For accessing the C&C server the bots performs DNS queries to identify the C&C server hosted by the DNS provider. Thus DNS monitoring helps in detection of DNS traffic anomalies. This approach might give false positives. another setback for this approach is the high processing time necessary to monitor a large scale of network. DNS based detection mostly concentrates on botnet tracking and understanding botnet characteristics and technology and less on detection of botnet infection.

D. Mining Based Detection

This technique is mainly based on the identification of the botnet C&C traffic which is not an easy task. As botnets use familiar protocol like TCP, HTTP, IRC, FTP etc for C&C communication, the botnet traffic clashes with normal traffic. The data mining approaches includes machine learning, classification, clustering and correlation to detect the botnet C&C traffic. Use of data mining technique for botnet traffic detection can be seen in Botminer [14] which clusters similar communication and similar malicious traffic. Cross cluster correlation is then carried out to detect the hosts that share same patterns in communication and abnormal activities. Botminer is independent of botnet infrastructure and protocol. It can detect real world botnets which includes IRC, HTTP and P2P based botnets and also gives a very low false positiveness.

1.4 LifeCycle of a Botnet

Analysis of a botnet behavior can be done with its life cycle [15]. Botnet classification reflecting its lifecycle and current resilience techniques are discussed in [16]. The botnet lifecycle can be summarized into seven steps as depicted in the Figure 1.6 and briefly explained as follows:

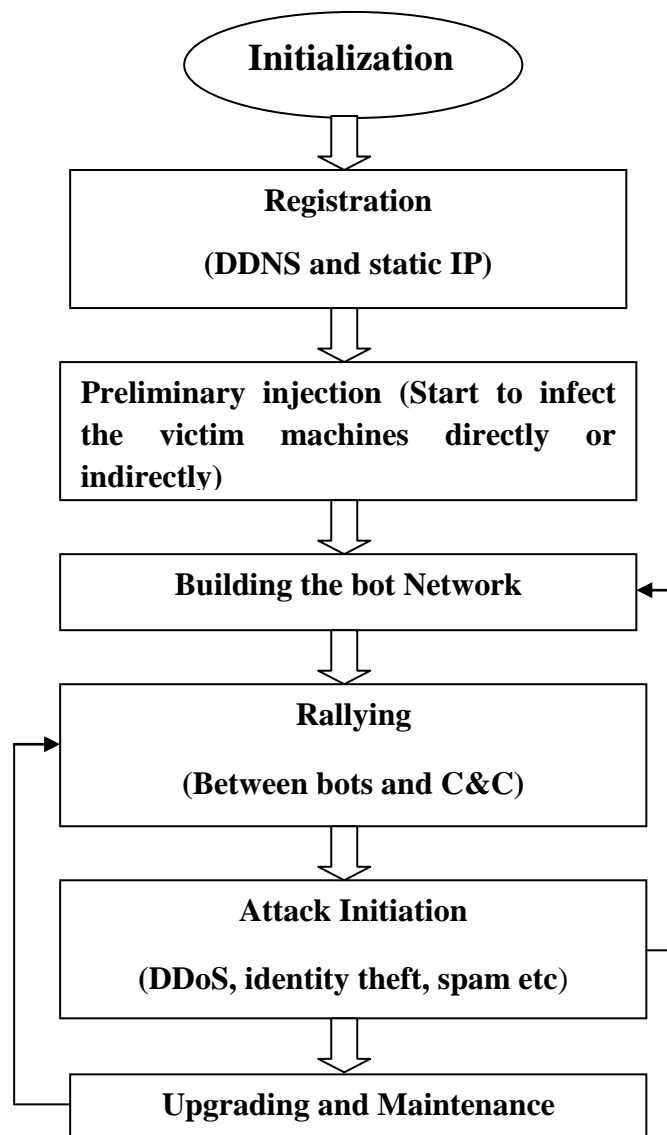


Figure 1.6 Botnet LifeCycle

1. The foremost step in the botnet lifecycle is ‘initialization’. In this step the communication is initialized by the botmaster by setting up the bot parameters.
2. The Distributed Domain Name System(DDNS) assigns a static IP address to carry out the registration process with the botmaster.
3. In the preliminary injection stage, regular infection is started in different forms like virus propagation, downloading of spam via email, running malicious attachments that are downloaded or infection by removable disk drivers.
4. After the preliminary stage, building of the bot network starts. Search is done by the infected systems and malware binaries from database are installed. The process of download occurs through HTTP, FTP or IRC protocols.
5. Rallying between the bots and the C&C starts the ‘Connection phase’. Whenever the bots restart, a connection is established between the bots and the botmaster and commands are sent and received.
6. Next comes the attack stage where the C&C sends commands to the bots to begin the malicious activities such as DDoS launch, search for loopholes or vulnerabilities in the computer system, identity theft, phishing attacks, software counterfeiting etc.
7. The last stage in the botnet lifecycle is the ‘Upgradation and Maintenance’ of the botnet stage. This process is performed by the botmaster so that all the bots stay up to date for all coordinated attacks in future. By updating, different detection techniques are being avoided, similar behaviours are avoided and addition of new features for connecting to other C&C channels.

1.5 Related Work

Recent history in network and cyber security has witnessed that much research has been done in botnet theory and its detection techniques. Honeypot research has also found equal importance as being one of the efficient detection mechanism for various types of cyber attacks including detection of botnets. This current thesis work is dedicated towards techniques for detection and mitigation of botnets and botmasters by deploying a low

interaction honeypot in a centralized C&C botnet architecture. A few of the related work are discussed below.

The HoneyNet project [17] is an ongoing open source project which basically aims at understanding honeynets to learn about bots. In this project, observation is done targeting the people who is in control of the bots and it is accomplished by setting up network of honeypots or honeynets. The paper discusses in depth about botnets, common attack mechanisms used and the people involved in carrying out the attack.

Niels Provos in his work [18], 'A virtual Honeypot Framework' brings out a framework for deploying a virtual honeypot namely Honeyd which can simulate computer systems at network level. The simulated machines seem to run on IP address which is not allocated in a particular network environment. Detection of network fingerprinting tools is done by simulation of networking stack of various operating system by the honeyd system. Honeyd proved to be very useful from security point of view. It could detect and disable worms, distract adversaries or prevent the spreading of spam emails. J.S Bhatia et al [19] also came up with an approach of botnet command detection using virtual honeypot. Researchers are coming up with efficient solutions using virtual honeypots/ virtual honeynets [20].

Evan Cooke et al [21] in their paper 'The Zombie Roundup' outlined the origin of the bots and their structures. They use data from the Internet motion sensor project and an experiment with a honeypot. The effectiveness of botnet detection is studied by monitoring of IRC communication flow. A botnet detection system is also implemented by utilizing advanced C&C systems by correlation of secondary detected data from various sources.

Michael Vrable, Justin Ma et al have built a honeyfarm system known as Potemkin which exploits virtual systems, extensively shares memory and resource binding in order to achieve the goal of better scalability of honeypots. Potemkin emulated around 64000 internet honeypots in tests run live using a limited number of physical servers.

Botnet initiated DDoS attack detection and mitigation has been described in [22] by Felix C. Freiling, Thorsten Holz and Georg Wicherski in their work 'Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks'.

Sivaprakasam.V and Nirmal sam.S has proposed a technique [23] to mitigate the effects of DDoS effects by collaborating a low interaction honeypot with a filtering technique named Firecol [24].

Other botnet detection techniques which does not particularly involve a honeypot implementation are Bothunters [25] which describes the detection with the help of cooperative behaviours, Botsniffer [13] which proposed a statistical approach to detect botnets based on their nature like spam distribution, binary download etc in a centralized architecture. Botminer [14] is an extended version of botsniffer which proposed a detection framework to do clustering on malicious activities and monitored C&C communication. The final result is generated by cross correlating them. Karasaridis et al. [26] built a detection mechanism to compare the distances between observed flow data and the IRC traffic flow model.

Detection by signatures involve Goebel et al. using regular expressions to symbolize sets of distrustful IRC names, and use of n-gram analysis and scoring systems for evaluation of the IRC names to determine whether a particular conversation is coming from a bot contaminated host . Detection also by observing attack behaviours have been proposed and implemented. Brodsky et al. [27] assumed that botnet tends to send a lots of spam within a short duration for detection of botnet generated spam. Also, Xie et al. [28] constructed a spam signature framework with the help of spam server traffic properties and spam payload.

1.6 Organization of the Thesis

This thesis has been divided into five chapters. A brief introduction of each chapter is given in this section.

Chapter 1 as discussed above gives us a broad understanding of the botnet threat and its detection and mitigation techniques. A detailed study is done and an overview is provided about the botnet phenomenon, its architecture, detection techniques and its lifecycle.

Chapter 2 introduces the Honeypot technology. It discusses about types of honeypots and classification according to its level of interaction. A detailed discussion is made about the importance of honeypots for network security. Also as per the main goal of this thesis, honeypot in detection of bots and botnet is also discussed. The chapter concludes with the recent advances and future trends in botnet technology.

Chapter 3 describes the proposed methodology of the current thesis work. It describes the scope and goal of the proposed approach. How a command and control architecture can be built with design structures and mechanism of the flow of control.

Chapter 4 gives a detailed understanding of the experimental setup and execution of the proposed approach. Creation of simple centralized C&C botnet architecture in an organizational private network has been explained in detail. Use of an external software called Hamachi to change the network so that the approach works in different networks is also discussed. Implementation of a low interaction Honeypot is explained along with the deployment of the honeypot. And finally detection of the bot/botmaster is carried out with the honeypot.

Chapter 5 discusses the experimental results that are obtained by implementing the proposed methodology and by making certain observations. This chapter justifies the proposed detection technique to be efficient and in contrast to the existing techniques.

Finally, **Chapter 6** concludes the thesis by discussing the overall contribution of the research in the context of related work in the area. In addition, this chapter also discusses the limitations of the approach and points to future research directions.

Chapter 2 HONEYPOTS AND THEIR TYPES

2.1 Introduction to Honeypots

Honeypots can be defined in many ways and history lays numerous definitions dependent on its usage and deployment. According to Lance Spitzner's book "Tracking Honeypots", honeypots can be defined as follows:

"A Honeypot is an information system resource whose value lies in monitoring unauthorized or illicit use of that resource"

By definition, any interaction with a honeypot is supposed to be unauthorized as it is practically of no production value. A honeypot is legally expected to get attacked, exploited and probed and in doing so it gives valuable information from security point of view. According to Wikipedia, "A honeypot is a trap set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems". So basically a honeypot is a real system acting as a decoy or trap when deployed in network system for the purpose of logging and studying various types of attacks in the world of internet. Honeypots can be applied as a solution to various security problems and one of the many solutions is tracking and detection of malicious botnets. It can log malicious activities in a compromised system which we call the bot or the zombie machine. Since there are plenty of configurations of honeypots, it is difficult to conclude what a particular honeypot does and how successful it is in meeting its objectives. For all requirements to be met including all legal, technical and privacy concerns, the purpose and goals of the honeypot to be deployed must be clearly stated in the security policy.

2.2 Types of Honeypots

Looking through the aim of honeypot [29], it can broadly be classified into two types namely research honeypots and production honeypots.

2.2.1 Research honeypots are mainly used in research, military operations and government organizations [30]. They can capture a large amount of information motivated towards capture of new threats and learn about blackhat techniques. Major objective of research honeypot is learning of protection measures and approaches. Security of an organization is not a primary concern in research honeypots. However, simulation of the whole operating system is possible with research honeypots which present the attacker with a familiar set of

vulnerabilities within the system. For example, in case of web related attacks, a default linux installation with apache can be done to observe the results. With a strict deployment, research honeypots can perform response tasks like trace-back. [31] The various uses of these type of honeypot includes security professionals trying to finds new attack techniques, government and law enforcement agencies looking for preventive measures in terms of predictions from research analysis. HoneyNet is a example of a research honeypot [32] [33]. HoneyNet research helped in revealing an increased trend in credit card fraud.

2.2.2 Production honeypots are mainly used as a protection system in an organization from external attacks. It is deployed inside a production network to improve the overall security. A lesser amount of information is captured compared to research honeypots. They can vary as per the level of interaction with the attacker. Production honeypots might pose some risks to the organizations existing security structure as honeypot features might misuse other systems without the knowledge of the network administrator. Production honeypots mainly aims at capturing data by emulating services and sending them to the network administrator. A honeypot is more efficient defensive mechanism than Intrusion Detection System and firewalls. As honeypots are of no production value any traffic coming towards it is malicious. Production honeypots also helps in noise reduction for malicious activity with lesser false positives. According to Bruce Schneier's model , honeypot security phases are primarily classified as prevention, detection and response.

2.2.2.1 Prevention

The foremost thing to consider in building any security model is prevention i.e preventing the system from being hacked. This can be done in many ways like using firewalls to control network traffic and adding some rules to stop or allow it, using authenticated access, digital certificates, password protection etc. Use of encryption algorithms are also seen for message protection . Using honeypots in a company network can provide prevention in many ways. A hacker aware of such honeypots will be scared or confused. Thus use of honeypots can help in prevention of any security breaches in an organization.

2.2.2.2 Detection

In cases where prevention did not work and a system is compromised by an intruder, there are ways for detecting those attacks. Network Intrusion Detection Systems is one of such detection solution. This type of solutions help users to know if their system or network is

breached or attacked but it cannot prevent the intruders to get into the system. A valuable monitoring technique at this point is deployment of honeypots.

2.2.2.3 Response

At response stage, it is quite certain that a system has been attacked and a response has to be generated. Forensic investigation begins at this stage. When an intruder compromises a system, some traces are left behind by them. With the help of appropriate data as clue, results can be derived as to what might have happened and how. Honeypots has the provision of logging files and information which are valuable for later investigation.

2.2.3 Honeynets

The idea of Honeynet comes from setting up a network of honeypots [32]. In a classical method, a single honeypot is deployed in a production network. But it is possible that more than one honeypot is deployed each being a standalone solution. A honeynet deployment requires a honeypot and a honeywall. The intruder can access a honeypot which is a real operating system and perform attack operations such as launching a denial of service attack. For risk reduction, a firewall is configured on the honeywall so that outbound connections are limited. Thus the production network remains completely unaccessible. The Honeywall can even maintain an Intrusion Detection System to monitor and record the packets going and coming towards the honeypot.

According to the Honeynet project, there are two honeynet architectures: First Generation (Gen-I) and Second Generation(Gen-II) [34] . Blackhats can easily discover the Gen-I types and they are easy to fingerprint. Due to lack of sensor on the honeypot operating system, the activities on the host are not stored separately and can be removed by the hacker. A honetnet can be accessed easily by a layer-3 firewall.

Gen-II honeynets are more advanced and not easily detectable. They can record events in the host and recording of keyboard strokes is possible even if the message used by the attacker is in encrypted form.

Figure 2.1 below shows a diagram which depicts a Honeynet setup with four honeypots. Acting in a bridge-mode, the honeywall does the functions same as the switches. This was, the honeywall is logically connected to the network by keeping the honeynet in the same address range.

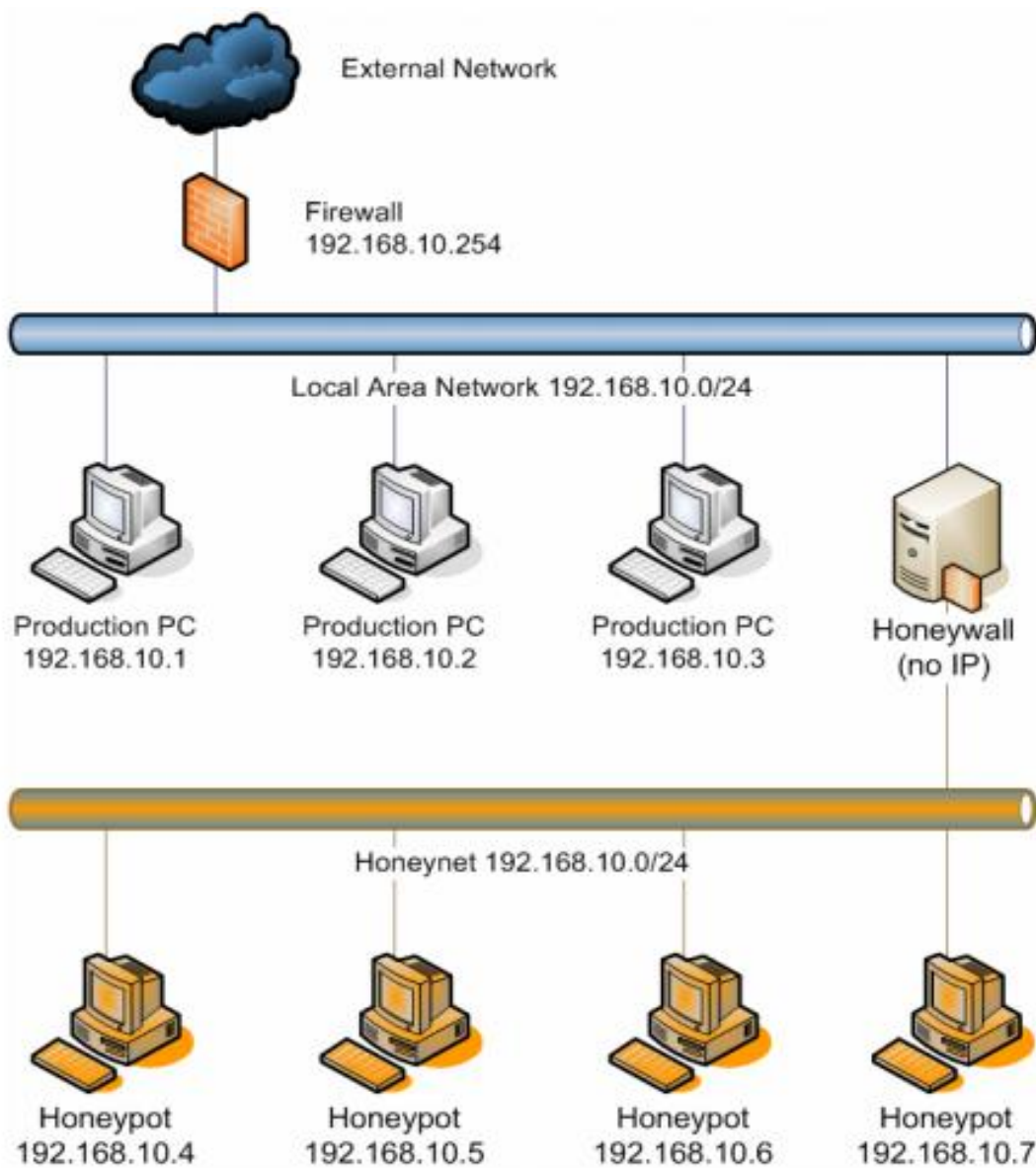


Figure 2.1 Honeynet Setup

2.3 Level of Interaction

Honeypots can be classified into three types according to their level of interaction with the attacker.

2.3.1 Low-interaction Honeypots

A low-interaction honeypot [35] can emulate a limited number of network services such that an intruder cannot perform any specific action [36]. They can be primarily used for detection

and logging purpose and work as production honeypots. Sometimes they also respond to some login attempts. An attacker can have access to a limited services and the underlying operating system is not hindered. Hence low interaction honeypots provide a secure solution promoting very less risk to the underlying environment. They are also successfully implemented in capturing and analyzing malicious packets in VoIP environment [37]. Example of low interaction honeypot is Honeyd.

2.3.2 Medium-interaction Honeypots

Medium-interaction Honeypots can emulate full services i.e they can emulate behaviour of web servers or operating systems with the primary purpose of detection as production honeypots. They are also used as an application on the host operating system like the low interaction honeypots and only emulated services are presented to the users. They differ from the low interaction ones in terms of chance of failure making them more vulnerable to attacks. Nepenthes [38], Honeytrap, Honeyfarm [39] etc are some examples of medium interaction honeypots. Nepenthes honeypot was used in malware collection along with anti virus scanning in [40].

2.3.3 High-interaction Honeypots

High-interaction honeypots are the most interactive amongst the three. Emulation of a full operating system or installing a real operating system or server is done in addition to monitoring systems. The primary purpose of high interaction honeypots are focused towards research although they can use as production honeypots also. The vulnerabilities are higher compared to low and medium interaction honeypots. An intruder can benefit highly by using the compromised systems to perform attacks in the network, launch a DDos etc. SQL injection analysis can be performed with the help of high interaction honeypots [41]. A good example of high interaction honeypot is Honeywall.

2.4 Honeypots in Network Security

In the field of network security, honeypots does the work of turning the table for both attackers and security professionals. In the world of honeypots, security breaches are exposed on purpose and various internet threats are invited. The primary purpose of honeypots is detection and learning from various types of attacks and using those information to improve existing security infrastructure. Unknown security holes and network vulnerabilities can be analysed using the information gathered from honeypots and thus protected. Depending on

the deployment of honeypots, security holes can be observed and analyzed. Information of attackers who gained access to the network can also be studied and their tactics can be noted.

Earlier network monitoring techniques like Intrusion detection and prevention systems used passive network traffic monitoring based on specific patterns. Such methods give false positive errors due to mismatch in patterns and sometimes even false negative errors on actual attacks. But on a honeypot, as the system is of no production value, every packet is malicious clearly stating that every device connected to a honeypot is either wrongly configured or with an evil intention. Hence attack detection becomes an obvious scenario in honeypot environment. A honeypot can be deployed in a network within or outside the firewall. These are the most vulnerable places from where attackers try to access the system. Honeypots has application in various fields, research honeypots can be used in education , honeypots in internet to measure actual attacks in the internet, hybrid honeypots or honeynets, honeypots with Intrusion Detection are few of the applications [42]. Research honeypots can particularly be used for the purpose of collecting malwares and monitoring of malicious activities [43]. A very effective solution came up when Koniaris et al depoyed two research honeypots where one behaved as a malware collector and the other acted like a decoy system which logs every malicious actions from the attackers. The system have been kept online for a long time to collect the information and detailed study has been done.

A good amount of research has been focused towards visualizing attacks on networks and computer systems by the information security community. A visualization tool for the popular low interaction honeypot, Nepenthes was presented by J Blasco. Nepenthes honeypot here acted as the malware collector which could simulate vulnerable services that are target to malwares so that binaries corresponding to worms are safely captured [44]. The honeypot Dionaea was also used in a visualization project named carniwwwhore which was designed in the form of a web interface for one of Dionaea's databases and a recent development in terms of utility was also seen for the Nepenthes platform [45].

2.5 Honeypots for detection of bots and botnets

Detection of botnets is a multiple step operation. The first step is to collect some information about the existing botnet which can be done with the help of honeypots or by studying the captured malware. For example the honeypot nepenthes automatically captures malicious packets and incoming traffic [40]. This way botnet related information can be analysed and

studied. With the use of a proper honeynet /honeypot architecture malwares and worms can be detected.

Bot and Botnet behaviour can be classified through bot commands as these commands corresponds to a particular action to be executed by a bot program [46]. Bot commands can be of different categories like general commands which are invoked by intruders to manipulate the botnet [47]. Examples of such commands are acquiring a nickname for the bot with nick command, terminating operation with terminate command etc. Second category is the Host Control Commands which are used in obtaining details of the host and causing some abnormal actions to the host. Examples are execute command to for executing an application. A third category is of network control commands which are purposed to extract information out of the host network and manage the behaviour of the network. Examples are net info, scan etc. The fourth category is of those commands which are meant to launch attack on victim machines. Examples include DDos, email spam etc. Table 2.1 below gives the group classification of the bot commands.

General Commands	Host Control Commands	Network Control Commands	Attack Commands
login/logout, reconnect, id, alias, action, join, part, privmsg, mode, cmdlist, about/version, disconnect, nick, rndnick, status, quit	remove/die, clone, open, delete, sysinfo, shutdown, listprocess, passwords, killthread, killprocess, execute, sendkey/getcdkey, keylogger, threads, opencmd	server, netinfo, download, update, dnsredirect, httpd/ httpserver scan, visit	synflood, updflood httpflood, pingflood email spam

Table 2.1 Classification of Botnet Commands

Chapter 3 PROPOSED METHODOLOGY

3.1 Scope of the Methodology and Main Goals

As discussed in the previous chapters there are two major approaches in detecting a botnet. One is setting up a honeypot/ honeynets and other being passive network traffic monitoring and analysis. The main aim of this thesis is detection of botnet command and control through deployment of a low interaction honeypot. In order to achieve that goal, a botnet C&C environment had to be build which will be in direct contact with the botherder [48]. A centralized architecture was chosen particularly as it is easy to deploy and also it is beneficial in terms of security perspective due to its single point of failure feature. The choice of low interaction honeypot is also due to easy deployment facility and low risk involvement as it emulates services and do not have real services which prevents the attackers from fully taking over control of the machine. Taking all these requirements into consideration, an attempt has been made to create and implement a simple botnet dubbed environment. By creating a simple botnet from scratch, some techniques used in real botnets can be highlighted which can help in better understanding the threats imposed by botnets and hence come up with better mitigating techniques.

3.2 Implementation idea

The current work focuses on building a simple web based command and control(C&C) server with encryption and it is capable of the following tasks:

- ✓ Tracking of bots
- ✓ Receive reports from bots
- ✓ To give commands to the bots like sleep, spam, scan, start etc. Also a java based bot program is implemented which reports to the C&C
- ✓ Malicious traffic sniffing.
- ✓ Send spam/ malicious messages/packets as directed by the C&C

3.2.1 Command and Control Server Implementation

A domain has to be first secured in order to build the botnet C&C server. The environment needs to be built with windows host and Apache, MySQL and PHP deployed in it. The host is

connected to a dedicated network (LAN network) with a static IP address. A facade website deployment will help in hiding the botnet C&C which contains a MySQL database and a directory named botcandc deployed at the root of the web server. The directory botcandc consists of two main files namely connect.php and dbConnect.php. For reporting to the C&C server and receiving further orders, the bots will be connecting to the main script file, connect.php. The botmaster will use the script dbConnect.php to manage the botnet. The design of the botnet dubbed environment is depicted in the following diagram.

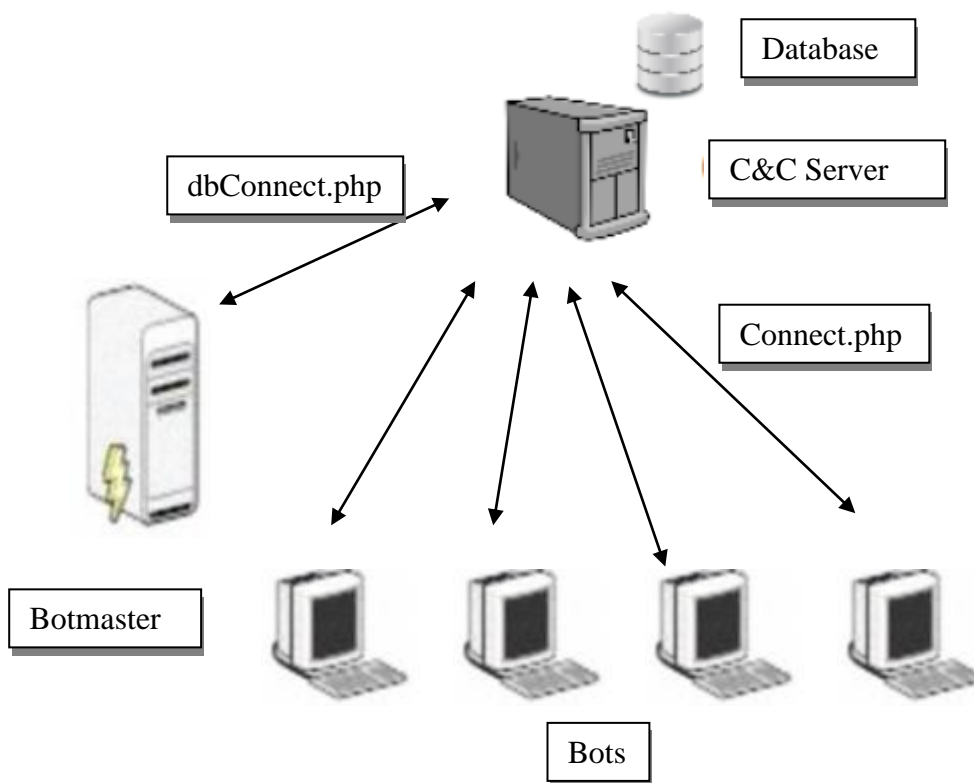


Figure 3.1 Design of Botnet C&C

The bots initiate the communication with the C&C Server via HTTP by sending data inside a POST which is the HTML method used in submitting data to be processed by a web server. The C&C will send the response in the form of simple commands for the bots to execute. When HTTP is used as the method of communication, detecting becomes a difficult task as HTTP traffic is too common for any POST requests to be lost among the valid traffic. That is the reason real botnets like *Torpig* and *Rustock* used HTTP as communication protocol. Also encrypted traffic makes it difficult for communication to be captured easily and also makes eavesdropping difficult. The bots are instructed to trust a self assigned SSL certificate which

provides protections to the transmitted data and the path of C&C files (botcandc/connect.php).

However a person or an attacker attempting to connect to the C&C server using HTTP cannot be stopped if he accepts the SSL certificate to be trusted. For these reasons a fake website need to be run in the C&C server. The encryption methodology will hide the traffic data but not the fact that communication is happening between with server. So it should be kept in mind that the botcandc directory remains hidden from anyone trying to access the fake website. This can be done by turning off indexes, website fingerprinting. Indexes can be turned off by editing the configuration files of the Apache web server.

3.2.2 Bots finding the C&C

After the command and control server had been built, the second step is to allow the bots to connect with the C&C. The URL with IP address of the host was hardcoded in the bot code:

```
ccInitialURL = "10.18.3.123/botcandc/";
```

Botmasters can use a Fully Qualified Domain Name(FQDN) instead of an IP address so that they can change the IP addresses of the C&C that has been compromised. Sophisticated botnets uses even more complex algorithms. Use of domain flux which uses a domain generation algorithm to find the C&C was used in the Torpig botnet [49]. Botmasters can even use ISPs for hosting websites, registrars for registering the domain names, and DNS/DynDNS providers for resolving the host names in the C&C.

3.2.3 The Bot model and transitions

The bots in their life cycle will go through various phases and transition which have been depicted in the following Figure 3.2 The information exchange between the bot machines and the C&C should be clear. Separate java classes need to be run in order to support the various functions to be carried out. A few of the classes will be discussed in the next chapter.

Bot going through various transitional stages:

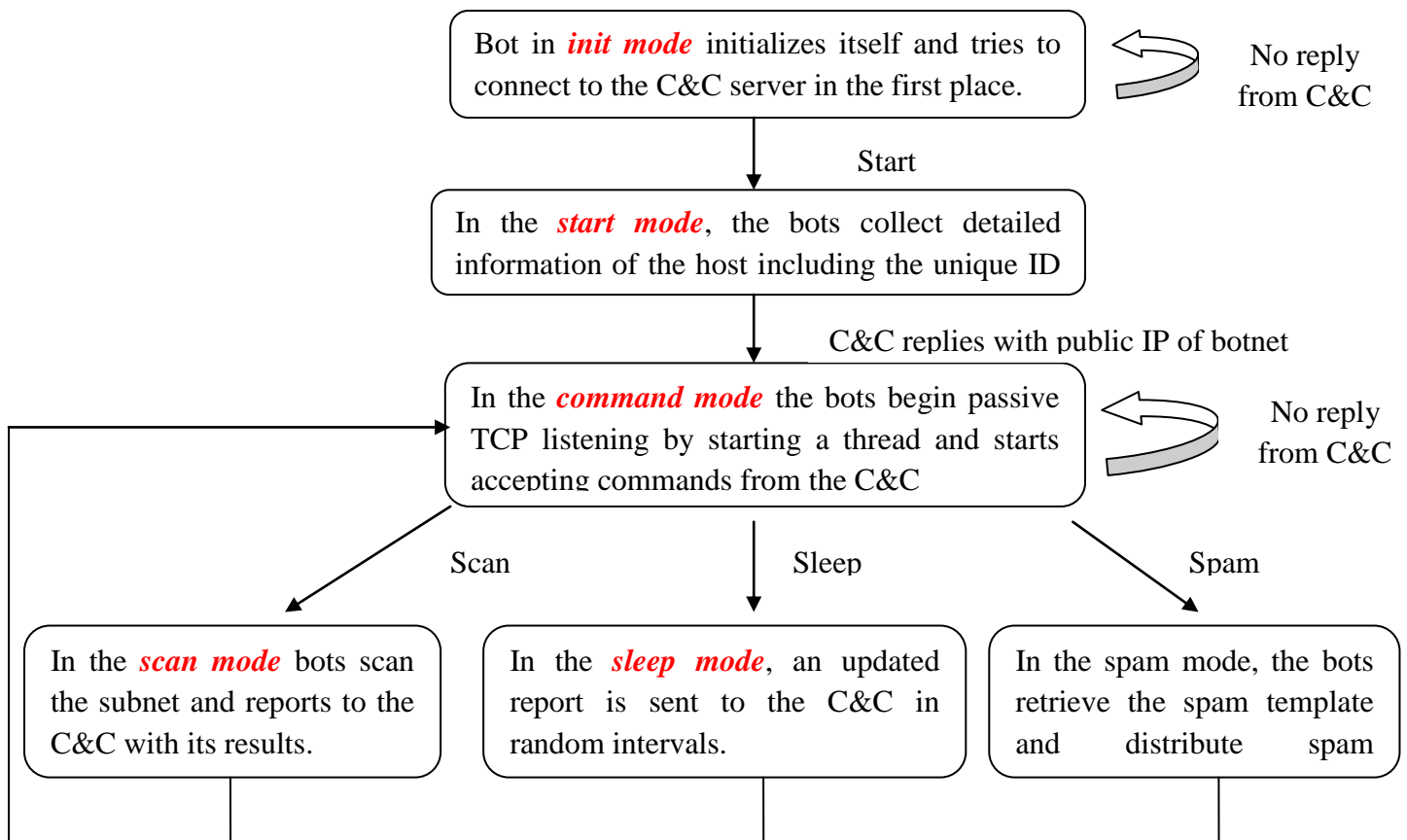


Figure 3.2 Various Transitions of bots

3.2.4 Bot reporting C&C

The bots, after identifying the C&C ensures that the commands are transmitted in a secure manner. The encrypted POST does this task by asking the bots to trust the SSL certificate of the C&C server. This is done in the method `setupTrust()` in the class `cc_Connector`.

```
Public void setupTrust()
{
Properties systemProps=System.getProperties();
systemProps.put("javax.net.ssl.trustStore","./jssecacerts");
System.setProperties(systemProps);
}
```

The file which is used as a trust store is named `jssecacerts` which consists of the public part of the SSL certificate used by the C&C web server. Implementation of a shared password has to be done to ensure that only the particular bots can connect to the C&C. The password is set when the bot object is created.

The bot class has a parameter called the `sleepCycle` which helps the bots to connect with the C&C at random intervals. To avoid obviousness from specific patterns the parameter `sleepCycleRandomness` is also implemented in the bot class.

The POST methods used are built with an array. When the bots use POST to send any message a multidimensional array is created which holds the names of the parameters and their values. The class `cc_Connector` processes the data and encodes the parameter name and value pairs. These data is collected by the C&C through `connect.php` file. The C&C sends response in the form of one-word command eg `scan`, `sleep` etc and the bot requests for specific details.

For all the exchange that takes place between the bot and the C&C, the password submitted by the bot is authenticated and if found to be false, the bot will be misdirected to catch error pages.

3.2.5 Tracking and Detecting Bots

Much research has been focussed into tracking and detection of bots and botnets till now. However keeping up with the new and emerging techniques of hackers and intruders, this field is still an open platform for researchers to have a fair assessment of botnets' footprints. Tracking of botnets through honeypots has grabbed much attention from security point of view. It has been proven to be better than most other techniques of botnet detection. Clubbed with other techniques like anomaly detection, intrusion detection etc, efficient methods can come up for both detection and mitigation of threats posed from botnet attacks.

Chapter 4 EXPERIMENTAL SETUP AND IMPLEMENTATION

4.1 Botnet Implementation

To keep the design simple and understand better, the Command and Control (C&C) in this particular experimental setup resides with the botmaster machine itself. The architecture which has been implemented is shown in the following Figure 4.1. It consists of a server machine which is the botmaster and one or more client machines which are the compromised bots controlled by the C&C residing within the botmaster. Generally the botmaster and the bots dwell in the same network (LAN/Wifi). However with the help of router port forwarding it is possible for the C&C to control the bots residing in different networks too. The current work focus towards implementing the C&C architecture where the client and server are in different network.

Port Forwarding is a NAT (Network Address Translation) function that forwards a communication request from one IP address and Port number grouping to another during the traversal of the packets/ messages through a network gateway like a router or a firewall. In my case, I have used the LogMeIn Hamachi software to have the port forwarding done when a bot / client machine from a different ISP than my organization's private network attempts to connect to the botmaster(C&C) or vice versa. The Hamachi software is run in the background of the client or the server machine. When it runs, a public IP address is generated which can be accessed across a network gateway like the router or the firewall. For example if the software is run on the server/ botmasters machine, the SocketClient.java class is fed with that public IP address with the specified port number and thus connection is established between computers working on different networks. This feature can be further extended to build network in P2P platform and also in the cloud environment where distributed computing paradigm are used.

As explained in the botnet life cycle, a computer is said to be compromised when a botmaster attacks the computer with its binaries. When those binaries or executables are knowingly or unknowingly run by the users, the bot's code (malicious packet/ virus/spam) is downloaded in the computer and the bot machine is compromised. The direction of flow of the implementation is as shown in Figure 4.1

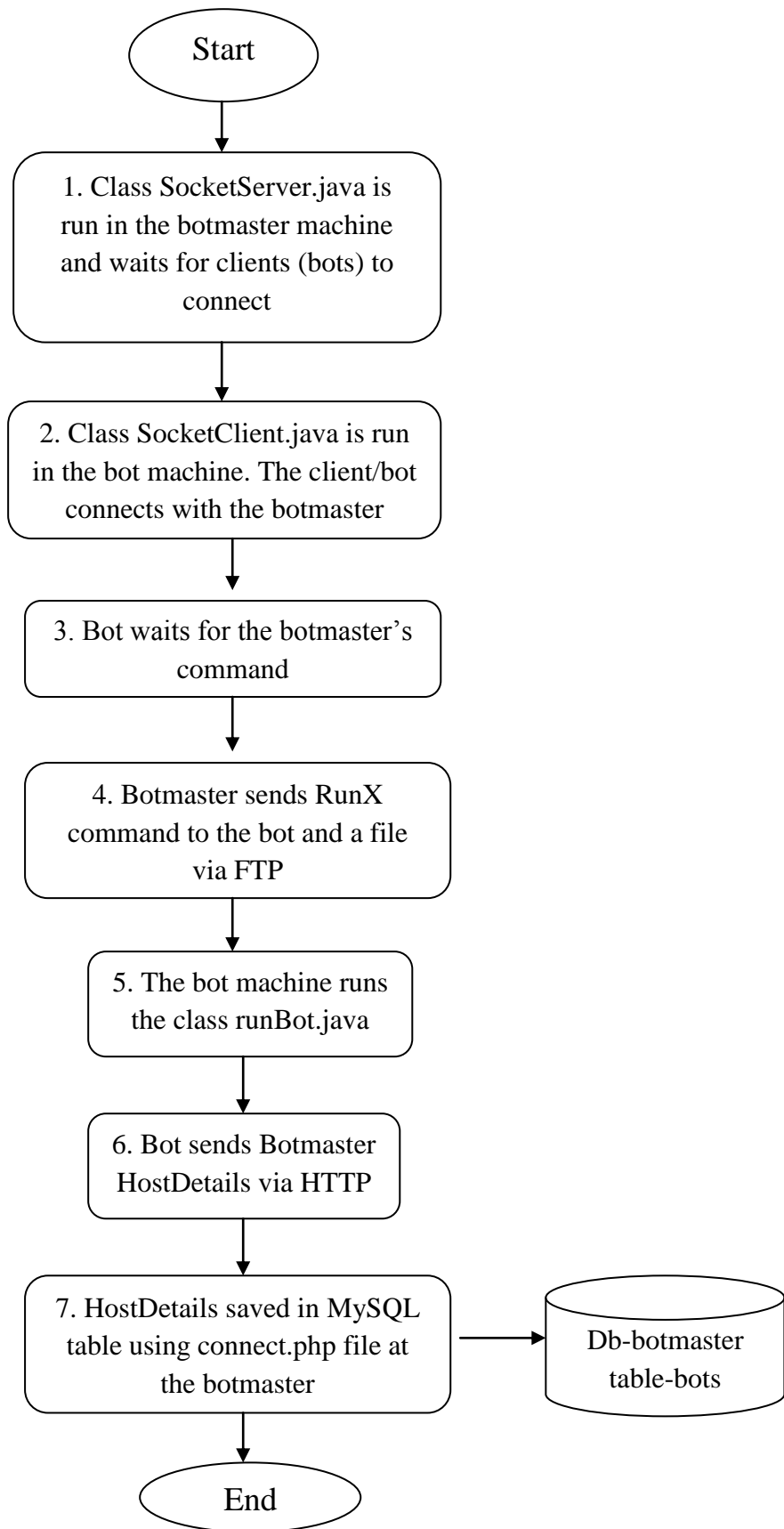


Figure 4.1 Botnet Implementation Flowchart

Programming Platform

Java is used as the programming language due to the following reasons-

- ✓ Java provides efficient networking modules to work with.
- ✓ The inbuilt libraries provided by java can be extensively used for easy development.
- ✓ Implementing network socket program is easy and stable.
- ✓ Java provides an extensive thread library for developing multi-threaded applications.

IDE(Integrated development Environment)

Eclipse IDE is used for the following reasons-

- ✓ It is open source software so comes free of cost.
- ✓ The features of the IDE include modularity, refactoring, code completion and package management.
- ✓ Eclipse also provides support for JavaDoc for documentation of source codes efficiently.

The machine with the botmaster runs a XAMPP server in its machine with Apache and MySQL module running at all times.

4.2 Communication Methodology

A network socket implementation is done using simple client-server architecture. The botmaster machine serves as the servers and the machines running the bot codes act as the clients. The socket is programmed to serve multiple clients at the same time and to have a bidirectional communication between the server and the clients. This is possible because of the multithreading feature of Java programming.

The series of steps carried out is as follows:

- ✓ Server package is deployed in the botmaster machine which contains the files SocketServer.java and SocketClientHandler.java. A new thread for each client is created using SocketClientHandler class. A botnet with a network of bots can be controlled in this manner by a botmaster. The server is always listening for connections with SocketServer.java always running.
- ✓ The machine containing the bot code is considered to be compromised with the class SocketClient.java being run on the machine. During execution of that class, the client

or the bot machine makes a connection with the botmaster hence creating a two way connection between the two socket end points.

- ✓ When SocketClient.java class is run on the bot machine, it asks for command from the botmaster via a message, 'Waiting for Command??'.
- ✓ Upon receiving the message, 'Waiting for Command??' from the bot/client machine, the botmaster machine, with the help of the class SocketClientHandler.java responds to individual clients by sending the command 'RunX' . Along with the command the botmaster can also send any file to the bot which can be inclusive of some encryption key or other malicious executables. Botmaster uses Telnet (via FTP) to transfer files.
- ✓ As soon as the bot machine receives the 'RunX' command, the class SocketClient.java executes the botRun() method of the class RunBot.java file.
- ✓ The main class which drives the entire bot code to carry out the tasks like scanning compromised machine, sending out details to the botmaster is RunBot.java. HostDetails are send from the bots to the botmaster with the help of HTTP by sending data inside a POST (the HTML method which is used for submitting data to be processed by a web server). The C&C residing in the botmaster will send the response in the form of simple commands for the bots to execute. When HTTP is used as the method of communication, detecting becomes a difficult task as HTTP traffic is too common for any POST requests to be lost among the valid traffic.

The Botmaster on receiving the information from the bots, stores them using PHP. The URL with the IP address of the host was hardcoded in the bot code. It typically looked like-

ccInitialURL= <http://192.168.137.239/botcandc/connect.php>

- ✓ The IP address 192.168.137.239 is the IP address of the server machine. Mine is a wifi network provided by our college. It will be changed according to the network to which the botmaster connects.
- ✓ botcandc is a folder in the root directory of the botmaster machine. The server used in this case is XAMPP server. So the root directory resides in the directory named htdocs in the XAMPP folder.
- ✓ Botcandc folder contains a file named connect.php which handles the POST data comprised of hostDetails and saving them in a MySQL table using basic SQL commands. Botcandc contains another file named dbConnect.php which makes the connection with the database.

- ✓ A database named botmaster is created in the XAMPP server and a MySQL table named bots is saved which contains the scanned information of the bots. We can later view the table via the XAMPP control panel.

The main classes which drive the class RunBot.java are briefly explained below. The actions of the bot machine is driven by the method botRun() present in RunBot.java.

Class Name	Purpose
Bot	It contains the features of each bot such as the current status, unique ID, sleep time, network specification
BotMain	The main class which is responsible for every bot actions
CC_Connector	It describes the connection of the brokers with the C&C via POSTS
CC_DataExchanger	Encoding of the data and request and response between C&C and data .
Tools	It contains some helper methods which are called from different functions.
HostDetails	It contains the detailed data regarding the host like OS information, uptime.

Table 4.1 Main Classes of the botnet Package

The various parameters which characterize the bots are as follows:

- ✓ Status- Current status of the bot (init)
- ✓ ccInitialURL- The initial URL given to the command and control
- ✓ sleepCycle- Duration in seconds to poll the C&C for instructions
- ✓ sleepCycleRandomness- Random behaviour in the sleep cycle
- ✓ id- It is the unique ID of the bot which is generated using computeMD5 function of Tools.java. This function receives hostname as the parameter and compute MD5 checksum of the same.

The MySQL table, 'bot' at Botmaster's C&C server-

botID	status	bot_Name	bot_osName	bot_osVersion	bot_osArch	bot_IP	botmaster_IP	Created	LastUpdated
8eb0a0956073b30b6f4d6ab99b4ae0ec	start	MYPC	Window 8.1	6	amd64	192.168.137.239	192.168.137.189	28-06-15 1:30	28-06-15 1:32

Table 4.2 Bots Table

Each row in the above bot table corresponds to details of the bots. The unique ID in each row is the botID. The table includes following columns-

- ✓ BotID- It is the unique id for each bot machine which is generated using MD5 encryption on the hostname of the particular bot machine. It acts as the primary key and does not accept duplicate values but updates the last updated field in the table if same botID is received in the database.
- ✓ Status- It refers to the mode of the bot machine. For example start or init.
- ✓ Hostname- It refers to the name of the bot machine.
- ✓ osName- The name of the Operating system of the bot machine
- ✓ osVersion- The operating system's version of the bot machine
- ✓ osArch- The operating system architecture of the bot machine
- ✓ hostIps- IP address of the infected bot
- ✓ sourceIP- IPaddress of the bot machine
- ✓ proxySourceIP- IP address of botmaster
- ✓ created- Date and time when the bot was created.
- ✓ LastUpdated- Date and time when bot was last updated.

4.3 Detection with Honeypot

Having set the botnet C&C architecture, now the detection of bots is done with deployment of honeypots. A low interaction honeypot named honeyRJ was implemented in a java based environment and eclipse as the IDE. *HoneyRJ was developed in the Spring of 2009 . It was a part of syllabus for a course project in CSE5715 named "Network Security at Washington University in St. Louis"*. As stated in previous sections a low interaction honeypot can serve one or more functionality protocols with limited services. HoneyRJ supported the following features:

- ✓ Multiple Protocols: Support for multiple protocols is provisioned by the application. The class which implements the interface can be included in the honeypot package and its logic will define the interaction of the honeypot with the clients.

- ✓ Can have an unlimited number of client connections: Due to its multi threaded design, the honeypot can make connections to any number of clients simultaneously. depicts the multithreaded design of the HoneyRJ.
- ✓ Logging : Each connection to the system with the honeypot has its logs recorded along with the information of the sent and received packets.
- ✓ Graphical interface: A user interface is created which allows a user to handle the application.
- ✓ The honeypot has also an inbuilt feature to prevent Denial of Service attacks. The two main features are *Connection timeout* and *Waiting period*. The connection timeout feature enables every connection to the honeypot to be closed after a configured timeout period of 2 minutes. It will force disconnect the connection if it remains idle for the timeout period or does not become idle at all. The waiting period feature will configure a period of 5 seconds between simultaneous connections of a protocol during which an attacker won't be able to establish new connections with the protocol.

4.4 Deployment of the Honeypot

The low interaction honeypot is deployed in the bot machines and it is run in the background of the client systems. When the bots try to connect to the C&C server in order to receive commands from the botmaster, the honeypot detects the connection and keeps a log of that connection as mentioned previously with the information derived from the bot machine. The GUI of the honeypot helps to identify how many connections are made simultaneously. The log files are created in the form of text documents in a specified directory with automatic updation. A user can even monitor the number of active connections from the log files. The format of a log file can be explained as follows:

```
*****Started at: Sat Jun 13 16:28:55 IST 2015*****
TIMESTAMP,SRC_IP:PRT,DST_IP:PRT,PACKET
```

The line after the header logs the details of each sent and received packets. In left to right order the information held are:

- Timestamp which says the time when the packet was sent or received.

- Source IP holds the IP address of the packet where it was sent from. For this scenario IP address of the bot machine for sent packet.
- Source Port gives the the port number used to send the packets.
- Destination IP is the IP address of the system where the honeypot is running.
- Destination Port gives the port number from which the packet was received.
- Packet means the string contained in the packet.

4.5 Key Implementation Features of the Honeypot used .

As included in the feature of HoneyRJ, the honeypot implemented in this case also has been programmed or modified to implement a multithreaded environment to give support for monitoring and interacting with more than one users simultaneously. This feature gives the provision of communication mechanism to be carried out by two clients on the same port with the help of different threads. A main thread continues to listen for new connections while new threads are used for handling connected clients.

In this particular work , the honeypot has been configured to operate in File Trasfer Protocol (FTP) which means that the honeypot can listen to and capture packets which are communicated between the clients (bots) and the server (botmaster) via FTP only.

The honeypot can be extended to provide support for additional protocols as inspired from HoneyRJ. It can happen with the help of the interface LIProtocol.java.

As the HoneyRJ logs the informations of connections, this honeypot also is designed with a provision of logging text files in an easily readable format for monitoring and analysis in future. Figure 4.2 shows the multithreaded design where two clients are connected at the same port.

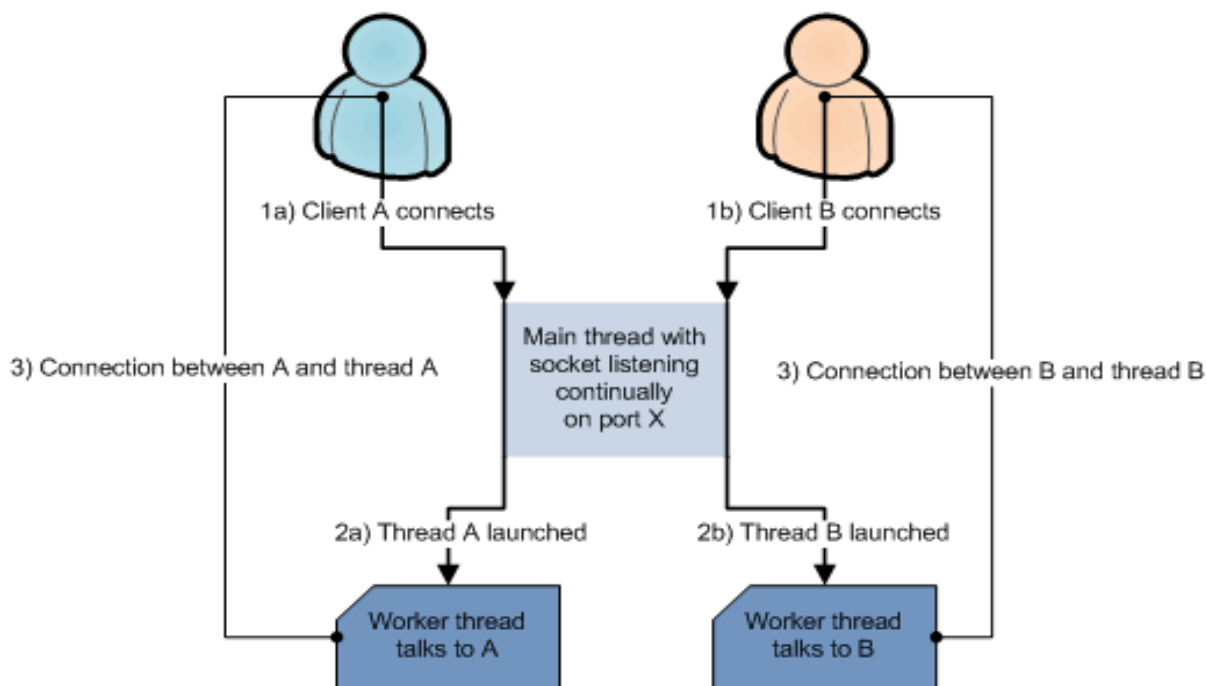


Figure 4.2 Multithreaded design showing two clients connected at the same port.

4.6 Honeypot Graphical User Interface

Typically the GUI of the low interaction honeypot looks like the one depicted in the Figure 4.3. The graphical user interface allows the user to control the application. This honeypot's GUI is created using Java's AWT (Abstract Window Toolkit) library. A user can start, stop, pause the application with the help of the GUI which are depicted with particular colours and hence change the listening mode for the implemented protocol (FTP in this case). Orange colour in the start button depicts an error.

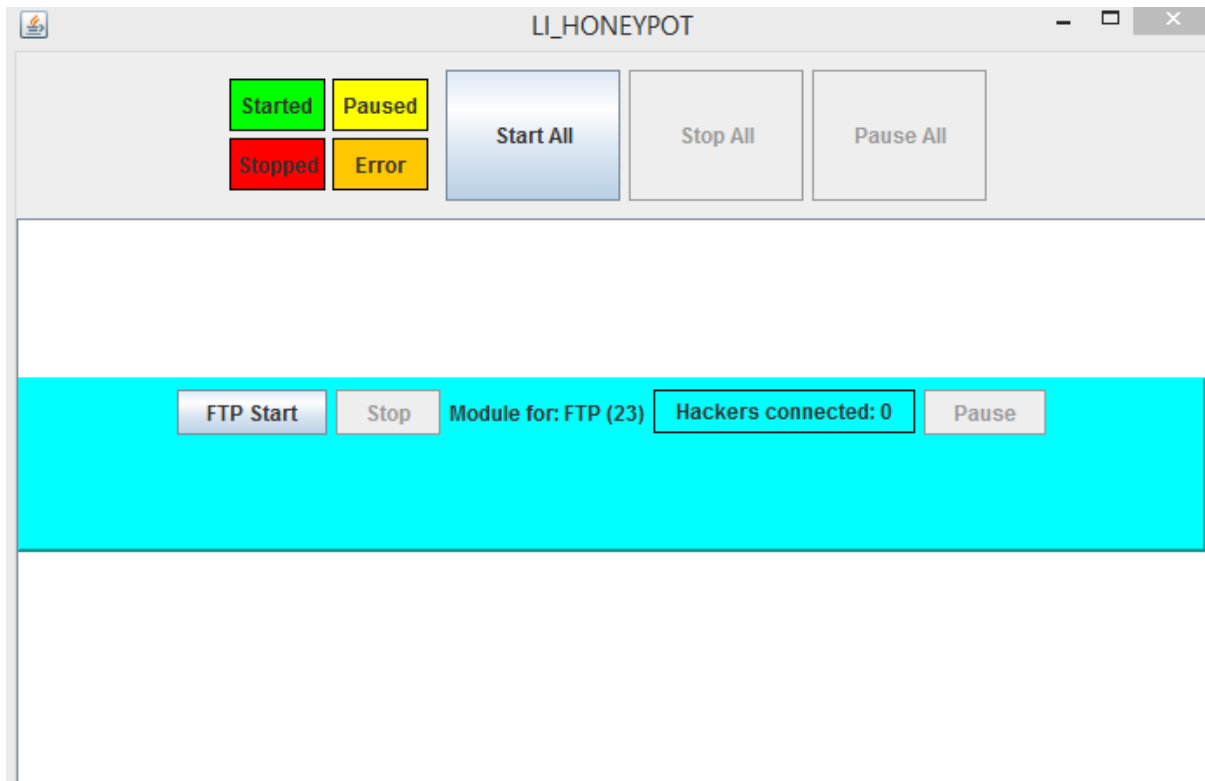


Figure 4.3 Honeypot GUI

4.7 Internal Mechanism of Honeypot Application

The various steps in the honeypot application describing the flow of events has been explained in this section. Whenever the application is launched and any protocol specific module is run like FTP start or stop some actions are triggered. The honeypot in this particular case aims towards tracking down and logging the details of any possible threat or attack from the botmaster with the help of a number of modules. These are low interaction modules which makes interaction with the protocol interface and the class which handles the threads that are launched to communicate with the clients. The honeypot application consists of mainly two classes namely honeyRJ.java which is the main application class and LIModule.java (Low Interaction Module) which provides support for communication with the protocol. Also there are two helper classes namely LIModuleThread.java and LIProtocol.java. A class in LIModule implements LIProtocol interface to give relationship logic with the clients connected. The LIModule also launches a LIModuleThread each time a client wishes to connect.

Three steps to describe the entire mechanism is described as follows:

- 1) Launch of the honeypot and initialization
- 2) Initialization of the LIModule
- 3) LIProtocol (FTP) interaction with client

4.7.1 Launching and Initializing of the Honeypot

The honeypot class consists of some specific modules and has a provision to manage modules by providing some services. As depicted in

Figure 4.4, the launch and initialization of the honeypot can be described in the following steps.

1. On launching the honeypot application, the honeypot class constructor is called.
2. A hashmap structure is created by the constructor which stores the implemented modules. The hashmap maps a port number to the LIModule to make sure that each port is loaded with a single module.
3. After initialization of the hashmap, next is creation of the logging directories. A reference is saved corresponding to each logging directory as a member variable so that it can be later passed to the added modules.
4. The Honeypot is ready to accept additional LIModules.
5. An instance of LIModules is launched and is sent to the RegisterService() method of the honeypot class wherein the hashmap adds up to make sure that the port is defined for that particular module. The method RegisterService() being called after addition of the hashmap, the logging directory gets access to it. By repetition of this process, other modules can be added.
6. The user then starts the module with the help of the GUI.
7. Once the module is started, the honeypot starts to actively listen for connections.

Application Launch Flow of the Low Interaction Honeypot:

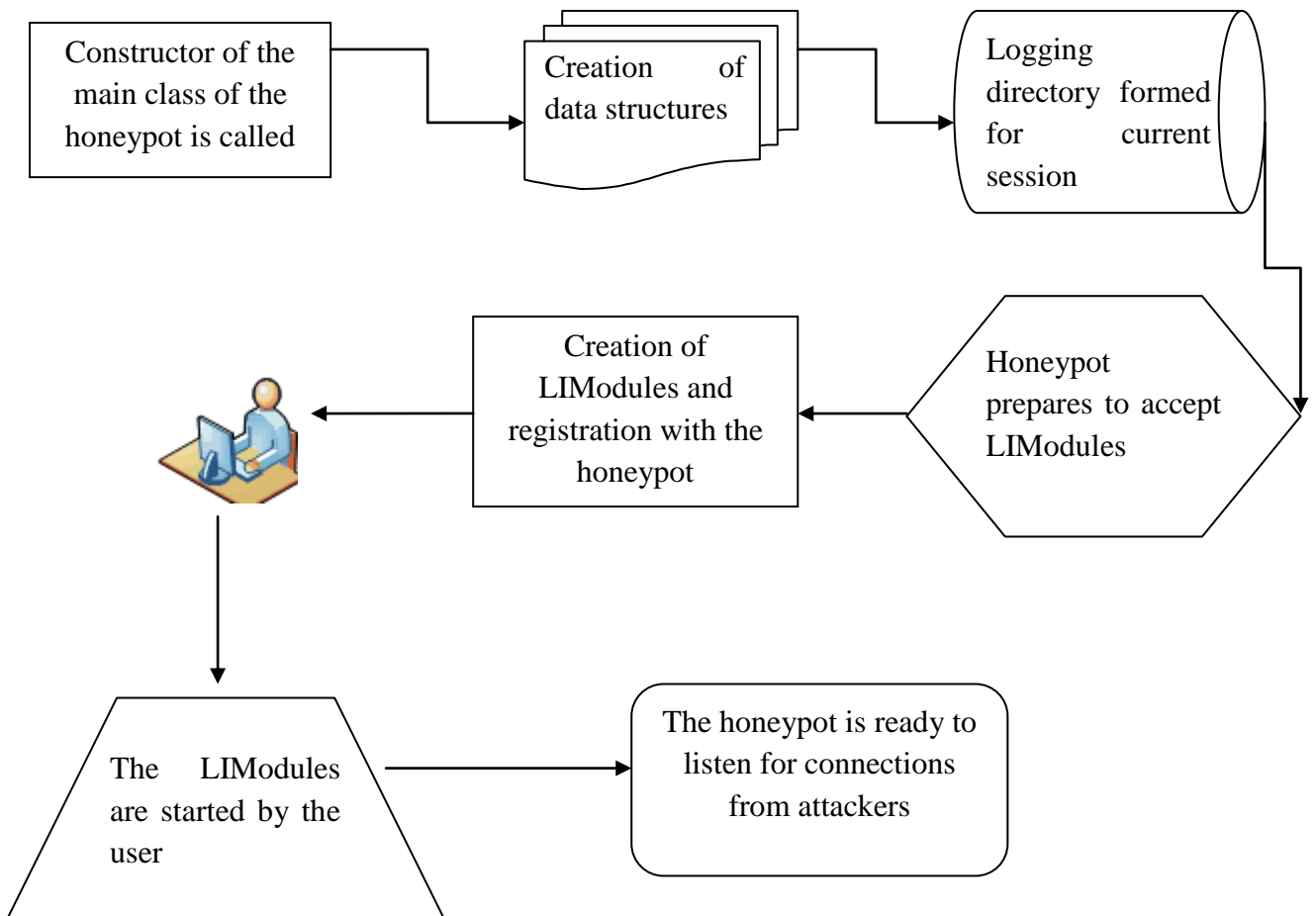


Figure 4.4 Honeypot Application Launch Flow

4.7.2 Initialization of the LIModule

This section provides an understanding of the steps that prevail after the LIModule is started. As explained in

Figure 4.5, the LIModule handles the implemented protocol in terms of communication and logs. The steps are described as follows:

1. For creation of the LIModule, the LIModule constructor is called and the initialized class implements the LIProtocol interface.
2. The constructor stores the LIProtocol class as member variables.

3. In this step, the LModule starts waiting for the honeypot class to register itself with the module. For this, the registerparent() method is being called.
4. After registration, a reference to the parent is stored by the LModule for accessing the logging directory of the parent in future.
5. The modules become ready for the user to start it.
6. The user then starts the module by clicking the 'start' button in the GUI corresponding to the specific protocol.
7. After start, the module is launched to a thread and a ServerSocket is created to listen on the port which has been specified by the implemented protocol. In this particular case, port 21 is used for the FTP protocol.
8. The module now listens for connection from clients. While a hacker or in this case, botmaster connects to the port, the LModule worker thread is launched with the connected socket.
9. The LModuleThread then connects with the botmaster in accordance with the LIProtocol and the LModule keeps listening for new connections.

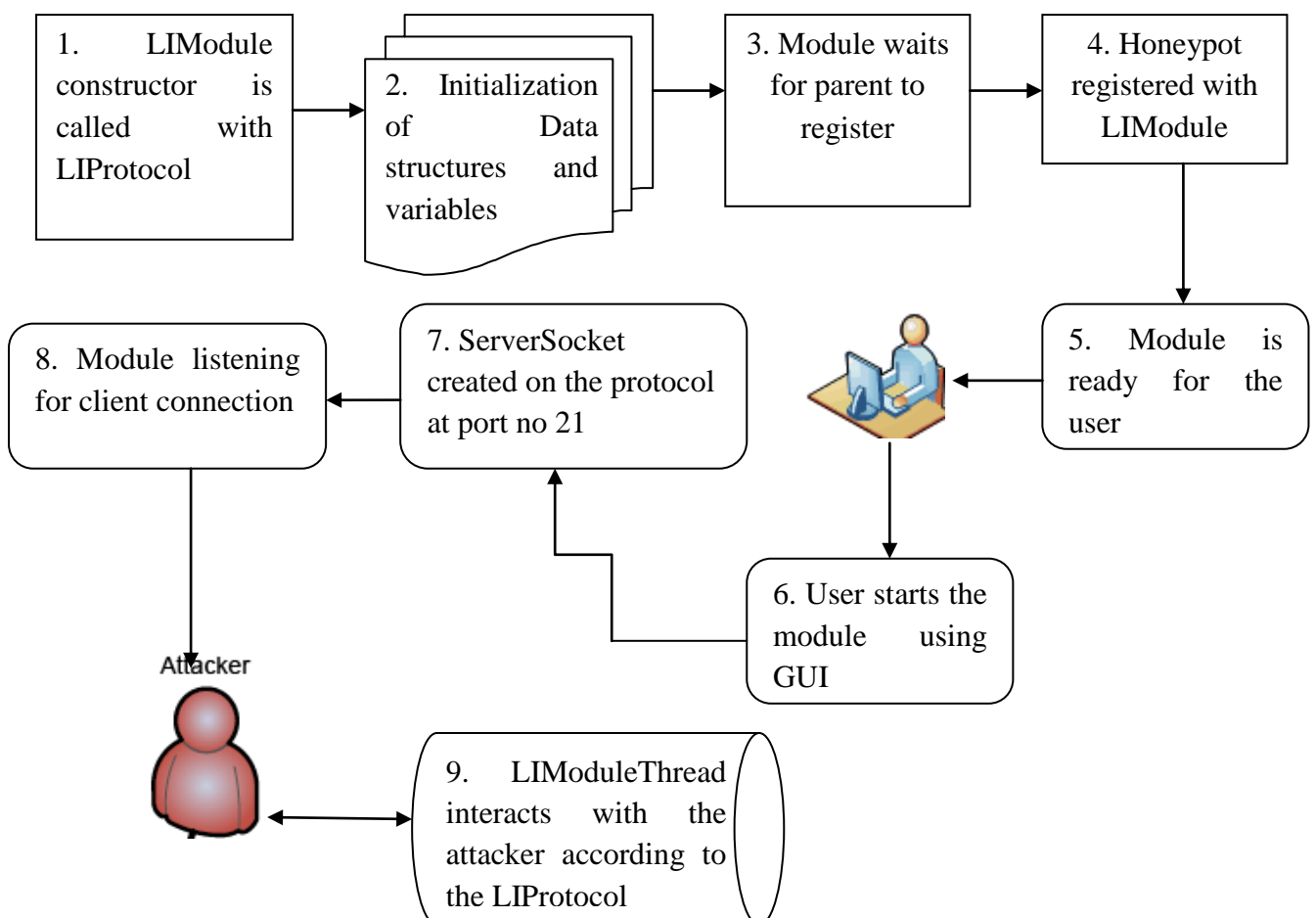


Figure 4.5 Flow of Events in LModule

4.7.3 LIProtocol (FTP protocol) interaction with Client

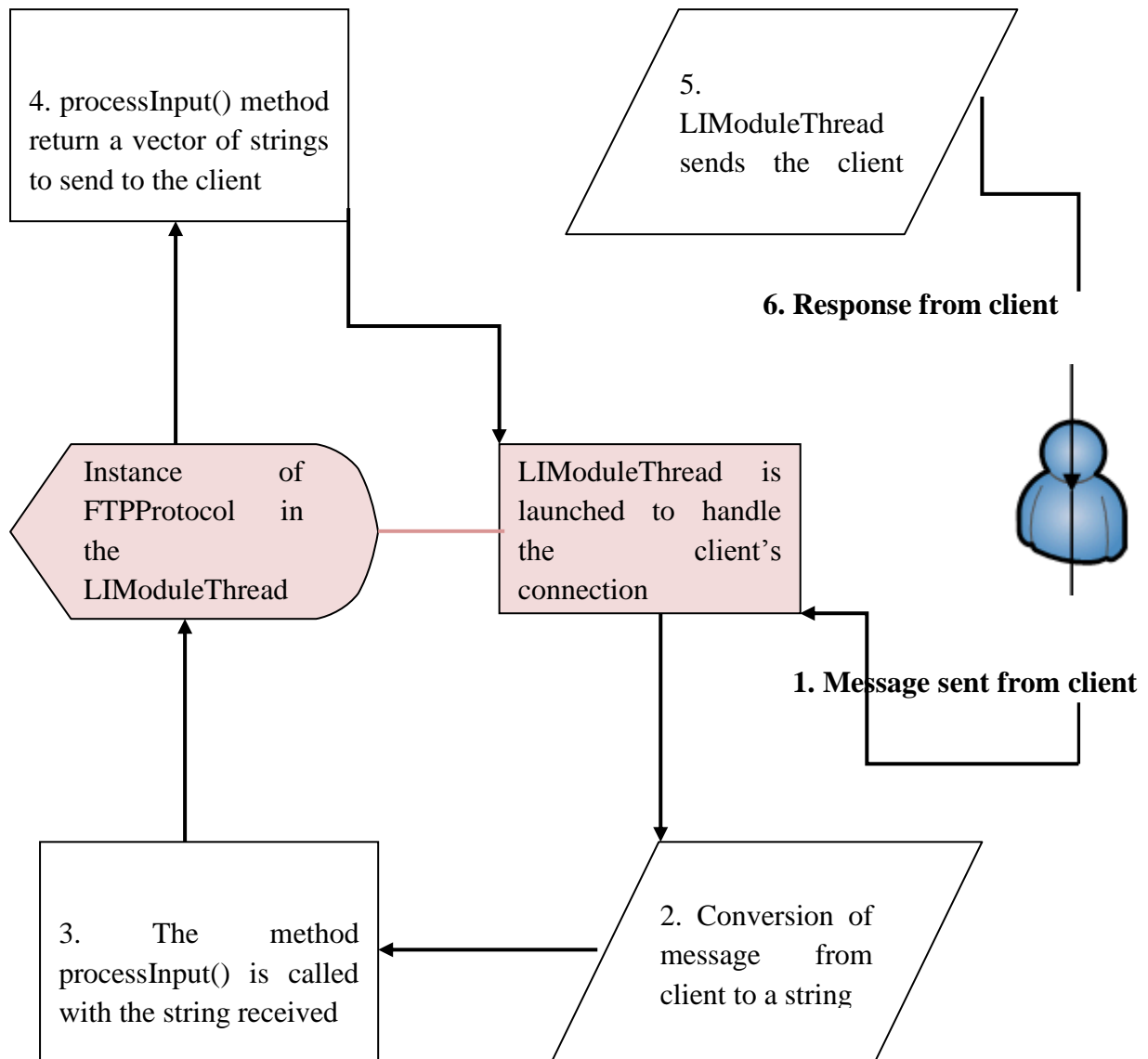


Figure 4.6 FTP communication between the client (Botmaster) and server (HoneyPot)

The LIProtocol's FTP implementation is shown in the above Figure 4.6. Of all the methods, the major work is carried out by the method processInput(). The other four methods are used to provide information about the protocol. LModuleThread creates an instance of the class and implements the LIProtocol interface while launching to handle client

connection. The process of flow of messages is shown in the figure 4.6 using the implemented protocol which is FTP in this case. Each packet received from the client on the socket is converted into a string object and passed as a parameter to the method processInput() when then processes the string and return its response to the client.

The honeypot implemented in this thesis work uses FTP protocol to process the commands and packets received from the clients(Botmaster in this case). This is accomplished with the help of the class FTPProtocol.java in the following manner.

- When the honeypot establishes connection with the botmaster, null message is received and the later returns the response as ‘Suspect is detected’. Another instance of message/packets sent from the botmasters end will confirm the presence of some malicious activity by the honeypot giving out the message ‘Malicious attack attempted’.
- As stated earlier a honeypot is a system which is solely dedicated towards attracting hackers and hence is of very less or no production value. So any connection or traffic directed towards it may be considered as malicious. To keep a check on the security of the network the botmaster’s machine has to be monitored regularly.

The honeypot as mentioned earlier logs every communication made to it. The logging is done in text file in the format mentioned in earlier section. The logs are saved in a local directory and can be processed and analyzed later for extracting useful information in detecting the botnet. The information directs towards the IP address of the botmaster which can be tracked down and necessary action can be taken to mitigate the malicious activities of the botmaster or the botherder. Some of such measures are stated as follows:

- ✓ The IP address detected in the honeypot to be harmful can be blocked from the network. This way an organizational private network can be secured from a known hacker.
- ✓ Any other system which are connected to each other via LAN/ Wifi or any other shared network can be intimated about the occurrence of probable attack by the hacker. So any kind of communication from that particular IP address can be restricted.
- ✓ On detection of the botmaster’s machine, other useful information can be extracted out of it. For example number of compromised hosts or resources, technologies used

by the botmaster, type of attack carried out , whether it is a DDoS / spam/ virus etc.
The botmaster's main motive behind the attack can also be tracked down.

As a whole we can say that detection with a honeypot can be an efficient mechanism to track down a botnet architecture along with its originator.

Chapter 5 EXPERIMENTAL OBSERVATION AND DISCUSSION

A java based environment is used for carrying out the client-server and the low interaction honeypot implementation. Experiments were carried out in eclipse IDE and the console results of the main classes are shown below which includes the results in terms of Botmaster (attacker/intruder), bots (machines which are compromised), details of the information that has been exchanged between the bots and the C&C, results from honeypot deployment, log files the honeypot stores in a local directory etc.

The current work puts particular emphasis on the client server architecture to work on different networks i.e network provided by different ISP's. This way the architecture can extend to the broader horizon of the internet and even to cloud architectures. To achieve this, a software to create VPN (Virtual Private Network) which provides extension of LAN-like networks to distributed teams has been kept running continuously in the background. The name of the software is LogMeIn Hamachi. It gives remote users provision of secure access to any private network/LAN including its resources from a centralized gateway like a router or a firewall. Just like router port forwarding, a client running on a network provided by my institution LAN network can connect to the server which is running on a network provided by Airtel ISP.

Client (bot) machine's console output

Figure 5.1 shows the console outputs of the bot (client) machine when the botmaster (server) is in the same LAN network and Figure 5.2 shows the console output of the same bot (client) machine when the server is running on a different network ie running on network provided by Airtel ISP. The public IP address 25.137.47.190 is provided by the hamachi software.

```
Console [X]
<terminated> SocketClient (1) [Java Application] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (25-Jun-2015 1:36:55 pm)
Attempting to connect to 10.60.4.156:21
Connection Established
RESPONSE FROM SERVER:RunX
New bot object created
C&C told me to start
Determining botID
This bot UID = 8eb0a0956073b30b6f4d6ab99b4ae0ec
Getting bot details
botpwd=12345678abcdefgh
status=start
botID=8eb0a0956073b30b6f4d6ab99b4ae0ec
hostName=MYPC
osName=Windows 8.1
osVersion=6.3
osArch=amd64
hostIps=192.168.243.1 10.60.2.33
ErrorC&C told me to sleep for now
I will now sleep for 10 seconds.
```

The client with ip 10.60.2.33 is trying to connect to server on the same network with ip address 10.60.4.156. Communication is made with response being received from server to run RunX

Figure 5.1 Client side console output (client-server in same network)

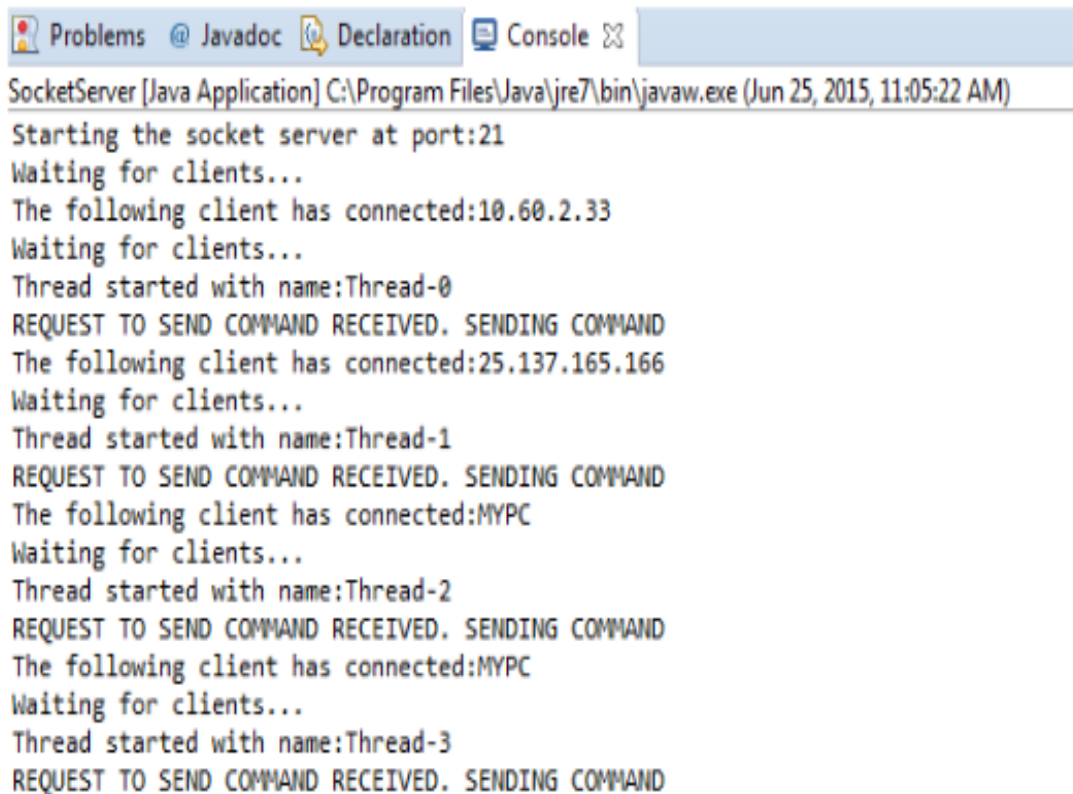
```
Console [X]
<terminated> SocketClient (1) [Java Application] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (25-Jun-2015 12:20:21 pm)
Attempting to connect to 25.137.47.190:21
Connection Established
RESPONSE FROM SERVER:RunX
New bot object created
C&C told me to start
Determining botID
This bot UID = 8eb0a0956073b30b6f4d6ab99b4ae0ec
Getting bot details
botpwd=12345678abcdefgh
status=start
botID=8eb0a0956073b30b6f4d6ab99b4ae0ec
hostName=MYPC
osName=Windows 8.1
osVersion=6.3
osArch=amd64
hostIps=192.168.243.1 10.60.2.33
C&C told me to sleep for now
I will now sleep for 14 seconds.
```

The client with ip 10.60.2.33 in LAN network is trying to connect to server on the network provided by Airtel ISP 25.137.47.190. Communication is made with response being received from

Figure 5.2 Client side console output(client-server in different network)

Server (botmaster) machine's console output

On running the class `ServerSocket.java`, the server starts a socket at port no 21 and it waits for clients to connects. Technically it starts listening for connections via FTP protocol at port number 21. On connection with any client commands are exchanged. Multiple threads can be created to connect multiple clients to the server.



```
SocketServer [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jun 25, 2015, 11:05:22 AM)
Starting the socket server at port:21
Waiting for clients...
The following client has connected:10.60.2.33
Waiting for clients...
Thread started with name:Thread-0
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
The following client has connected:25.137.165.166
Waiting for clients...
Thread started with name:Thread-1
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
The following client has connected:MYPC
Waiting for clients...
Thread started with name:Thread-2
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
The following client has connected:MYPC
Waiting for clients...
Thread started with name:Thread-3
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
```

Figure 5.3 shows the console output of the server program which displays thread number along with ip address of their respective clients. The thread name happens to be the host name itself. The XAMPP server running in the background runs a PHP script to extract information from the bot machine. A database is predefined to store the information as per the respective attributes.

Server side console output:


```

SocketServer [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jun 25, 2015, 11:05:22 AM)
Starting the socket server at port:21
Waiting for clients...
The following client has connected:10.60.2.33
Waiting for clients...
Thread started with name:Thread-0
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
The following client has connected:25.137.165.166
Waiting for clients...
Thread started with name:Thread-1
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
The following client has connected:MYPC
Waiting for clients...
Thread started with name:Thread-2
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND
The following client has connected:MYPC
Waiting for clients...
Thread started with name:Thread-3
REQUEST TO SEND COMMAND RECEIVED. SENDING COMMAND

```

Figure 5.3 Server side console output

Database at the server machine

botID	status	bot_Name	bot_osName	bot_osVersion	bot_osArch	bot_IP	botmaster_IP	Created	LastUpdated
8eb0a0956073b30b6f4d6ab99b4ae0ec	start	MYPC	Window 8.1	6	amd64	192.168.137.239	192.168.137.189	28-06-15 1:30	28-06-15 1:32
d432a25a6a7305505732b22d6bb2a6984	start	Purti-PC	Window 7	6.1	amd64	10.18.1.121	192.168.137.189	28-06-15 2:01	28-06-15 2:06

Figure 5.4 Bots Table at botmaster database

The information retrieved from the bots machine is stored in the table named bots created in the botmaster database. The database includes the details such as unique ID of the bot machine, status of the activity such as start, stopped, name of the bot machine, bot operating

system(OS) name, OS version, OS architecture, IP address of the bot machine, time of its creation and updation.

On establishment of a new client connection, the information of the bots are forwarded to the server in an array. The php script checks match between the botid and the id saved in the table. The LastUpdated table is updated id there is a match. SQL queries can be run on the table for further use.

GUI of the Low Interaction Honeypot

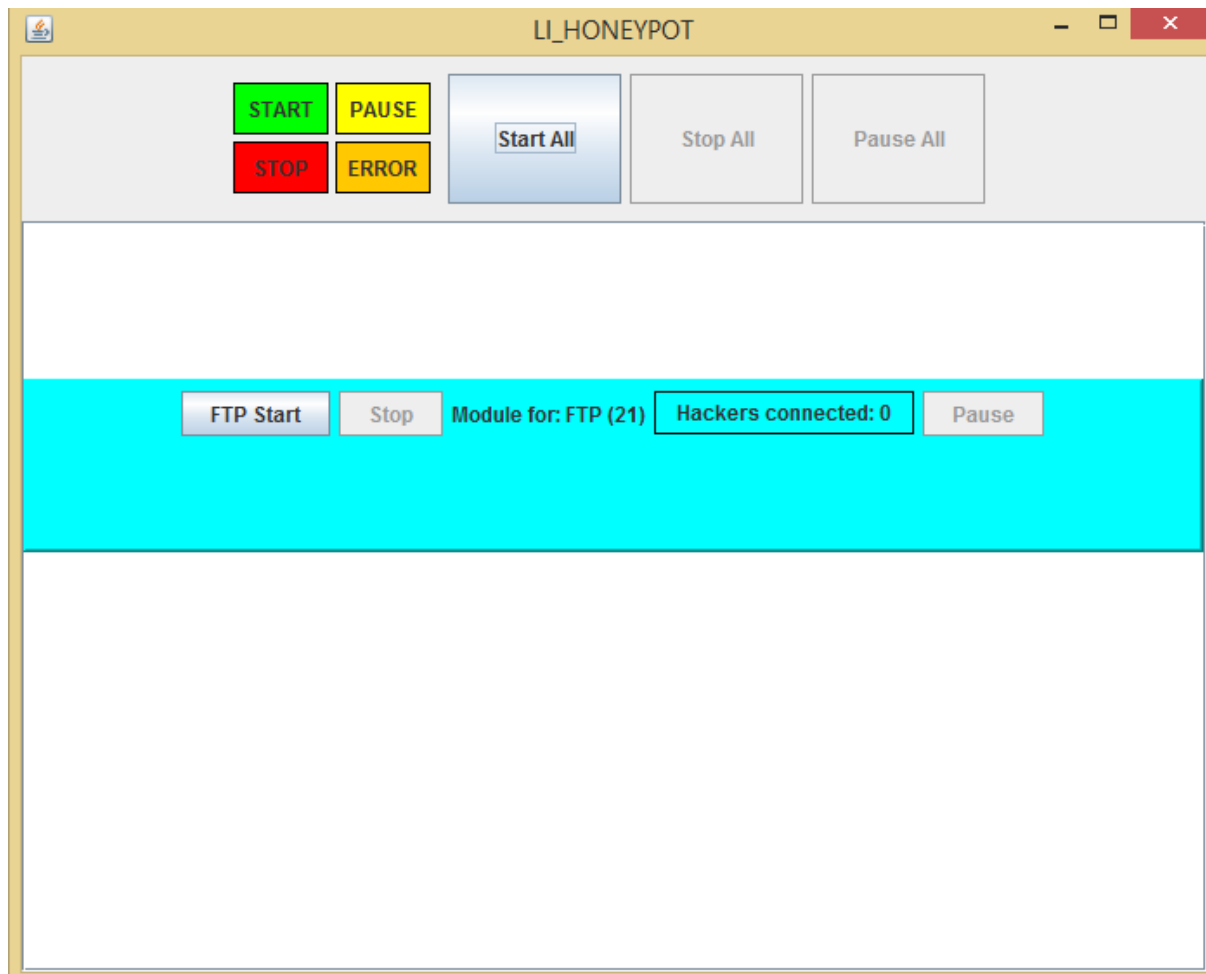


Figure 5.5 Graphical User Interface of the low interaction honeypot

The graphical user interface of the low interaction honeypot similar to HoneyRJ is shown in the Figure 5.5. In this particular honeypot FTP protocol is used as a communication mechanism and is listening at port number 21. As seen in figure, five states of the module are represented with five different colours. At the initial state, the module is in idle state (represented by blue colour) and honeypot is not started in this state.

The **START** state represented by green colour states the Running state of the Honeypot. On clicking it we allow the honeypot to run and start listening for connections at port 21.

The **STOP** state represented by red colour states that the honeypot has stopped listening for connections.

The **PAUSE** state with yellow colour describes pause state for the honeypot and orange colour will depict **ERROR** state.

Honeypot in START state with three clients / hackers connected

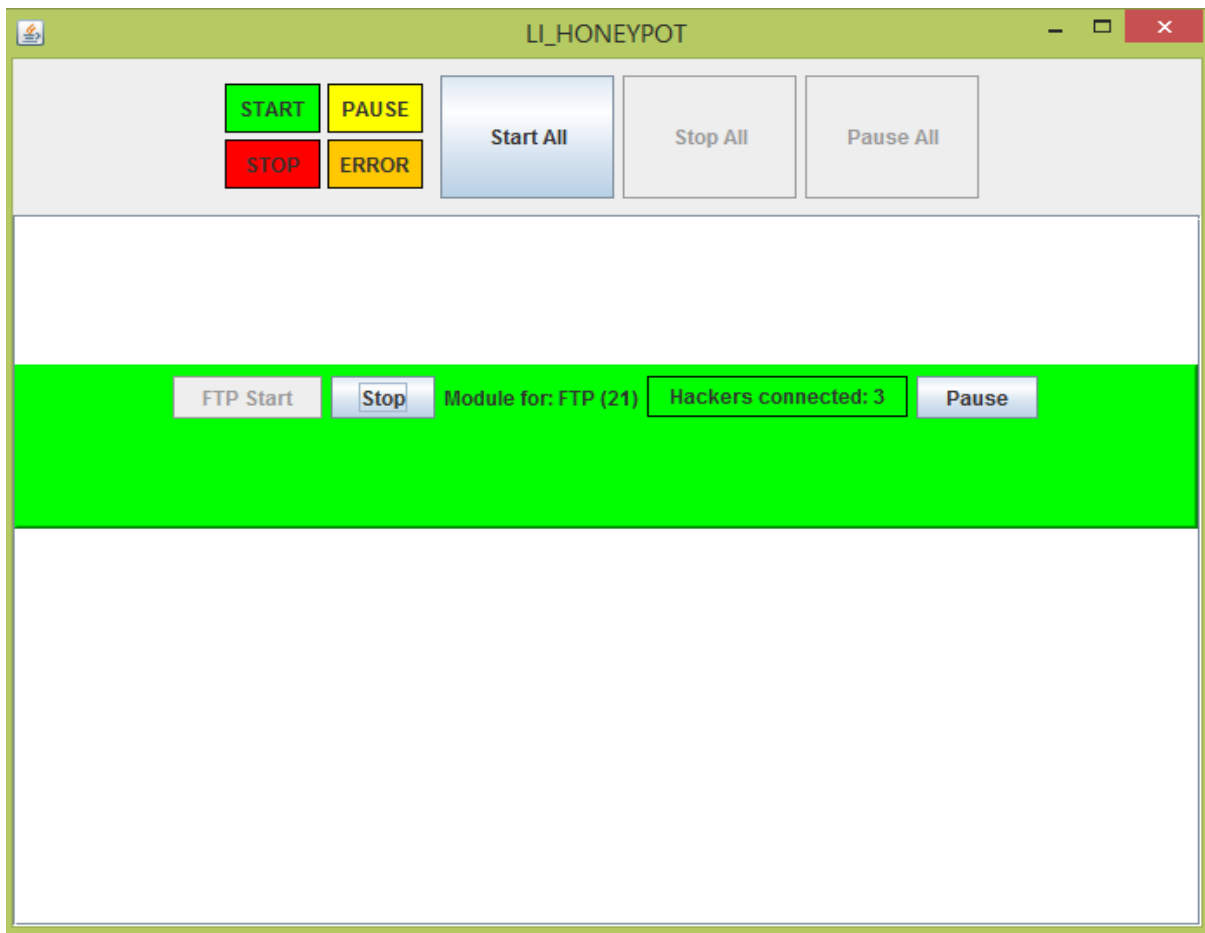


Figure 5.6 Honeypot GUI with client connected

While the honeypot is in running mode i.e when the start button is on, it is open to accept client connections. So whenever an attacker tries to perform any malicious activity via the protocol specified (i.e FTP), then the honeypot detects that connection and signals about the suspicious activity by showing an increment in the hacker count from 0 to 1. The count increases with increase in the number of clients or hackers. File Transfer Protocol has been extensively chosen because of the fact that it is a common scenario that hacking or any malicious activity is carried out mostly using FTP. Error state is observed with change to orange colour whenever the port 21 is used more than once and is in preoccupied state. This honeypot has the provision of connection time out and so after a particular timeperiod the hacker count automatically decrease and eventually comes down to 0.

Communication over Telnet between the bots and the botmaster.


A screenshot of a telnet window with a black background and white text. The text displayed is: 'Suspicious activity.', 'start', and 'Possible unknown malicious attack.'. The window has a yellow border and a vertical scrollbar on the right side.

Figure 5.7 Botmaster telnet window when bot is connected

The honeypot detects the attacker and sends a message to the server side i.e the botmaster that his attempt to connect has been suspected. It is indicated by a message displayed at the botmaster machine's telnet window, 'Suspect is Detected' as shown in the Figure 5.7. If the attacker continues to maintain the communication with another message like start, the honeypot will display the message 'Malicious activity attempted'. The flow of communication is in the following sequence:

1. The botmaster is warned of his suspicious message by the honeypot.
2. Botmaster continues to send commands to the clients to run a suspicious packet or download and run a harmful executable.
3. Indication of suspect detection on the telnet window by the honeypot.
4. The connection breaks in between the bot and the botmaster in order to save the user machine from any intended damage.

Suppose an attacker tries to launch a DDoS attack from the client machine with an intention to bring down the system. It often happens when an attacker becomes aware of the presence of honeypot . So in order to prevent itself from getting detected or its activities getting monitored, they try to crash the honeypot system. Such attacks often fails the security firewalls leaving the system vulnerable to unknown attacks.

Log files created by the honeypot saved in a local directory

```
Console [x]
HoneyRIMain (2) [Java Application] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (25-Jun-2015 10:29:15 pm)
F:\DCE\Thesis_work\Honeypot logs\honeypot_log
LIModule
LIModule
LIModule-run
LIModuleThread-Constructor
LogFile-constructor
loggingAddresses-
/192.168.137.239
/192.168.137.239
21 62190
LIModuleThread-run
LogFile-setBegin-writeToFile
*****

Started at: Thu Jun 25 22:29:52 IST 2015

TIMESTAMP, Source_IPAddress : Port Number, Destination_IPAddress : Port Number, PACKET (Messages exchanged)

whoTalksFirst- SVR_FIRST
ftpProtocol
Thu Jun 25 22:29:52 IST 2015,192.168.137.239:62190,192.168.137.239:21,Suspicious activity.

Monitoring interaction
Thu Jun 25 22:29:52 IST 2015,192.168.137.239:21,192.168.137.239:62190,COMMAND?

ftpProtocol
Thu Jun 25 22:29:52 IST 2015,192.168.137.239:62190,192.168.137.239:21,Possible unknown malicious attack.

LIModule-run
*****Protocol FTP TIMED OUT talking to /192.168.137.239 using local port 21, connection closed.*****

Stopped at: Thu Jun 25 22:31:53 IST 2015

*****
```

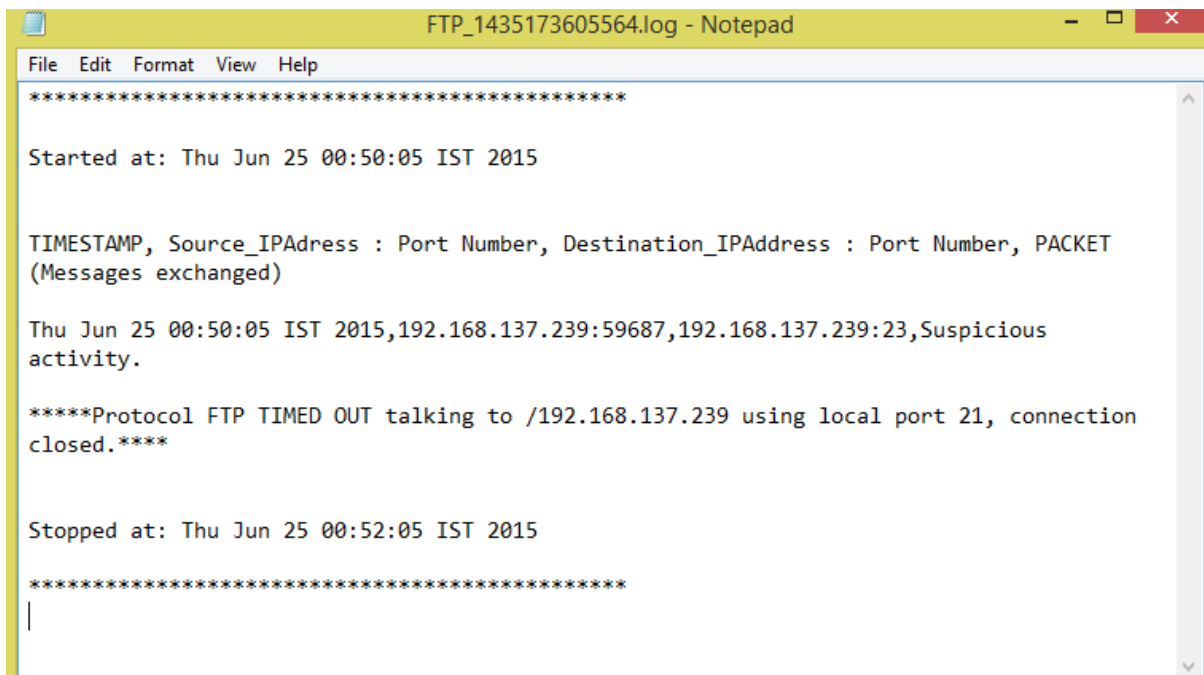
Figure 5.8 Console output of the log file created by honeypot

As mentioned in previous chapters, a log file has been created by the honeypot and stored in a local directory as assigned in one of the java classes. A typical log file looks like the ones shown in Figure 5.8 and Figure 5.9 displayed below.

Figure 5.8 shows the log file that outputs in the console window when the bot and the botmaster try to communicate. It starts with the line ‘Started at’ indicating the starting time and date of the communication.

Figure 5.9 is the log file saved as text format in one of the local directories to be observed and analysed for later purpose. It also has the same format which starts with line: started at followed by the corresponding timestamp once a connection is established.

The log file stores the source and destination IP address and corresponding port numbers followed by the information of packet exchange. Each of the sentences starts with the timestamp of respective event. By source and Destination, it means the source from which packets are sent and destination to which the packets are sent to. For example if the honeypot is deployed in the client(in this case the bot) machine and if the botmaster try to send any command to it, then the source will be the botmaster and the destination would be the bot.



```
FTP_1435173605564.log - Notepad
File Edit Format View Help
*****
Started at: Thu Jun 25 00:50:05 IST 2015

TIMESTAMP, Source_IPAddress : Port Number, Destination_IPAddress : Port Number, PACKET
(Messages exchanged)

Thu Jun 25 00:50:05 IST 2015,192.168.137.239:59687,192.168.137.239:23,Suspicious
activity.

****Protocol FTP TIMED OUT talking to /192.168.137.239 using local port 21, connection
closed.****

Stopped at: Thu Jun 25 00:52:05 IST 2015
*****
|
```

Figure 5.9 Log file saved in text format

Chapter 6 CONCLUSION AND FUTURE WORK

6.1 Conclusion

The threat posed by botnet attacks have been in an increasing trend since its inception. Recent attackers are mostly driven by the financial gain most common among the internet attackers. As the threats imposed keeps increasing, the security defenders are also in a continuous effort to come with efficient solutions and approaches to combat those threats. Although researches have put forward their best practices to come up with efficient detection mechanism, botnet research is still in infancy. Research in honeypot / honeynet and its deployment has a significant impact and value in the security community. But use of honeypots in detection and mitigation of botnets is still a novel concept. Keeping in mind that mitigating the ever increasing botnet protocols and structures is quite a challenge, an attempt has been made in this thesis to come up with an efficient solution for detecting bots and botnets with the help of honeypots.

Botnet detection has broadly been categorized into six types which includes detection by honeypots. Considering a few limitations in detection techniques like the signature based detection which can detect only known bots in the network, mining based detection not being real time and comparing the various detection mechanism [7], the honeypot mechanism is concluded to be one of the effective and promising mechanism. So in this thesis, a centralized botnet architecture is implemented with a low interaction honeypot deployed in the network for its detection.

Botnets commonly use the centralized architecture for its communication mechanism even though the P2P concept is also emerging. A centralized C&C structure is used for the bots to connect with the botmaster or the botherder in a network. This thesis implements a centralized botnet architecture with a server being the botmaster and different clients as the bots trying to communicate with each other. The client server architecture is implemented using java socket programming. The botmaster controls its network of bots by sending malicious commands and scope is provided for carrying out other attacks like DDoS, spam, phishing, sniffing etc. Then for preventing and mitigating the effects of the malicious attacks, the honeypot detection technique is also implemented to detect any hacker or the botmaster in this case which try to connet to the botnet or the bots. A low interaction honeypot inspired from the open source honeypot HoneyRJ is implemented also using a java based

environment. The honeypot does the detection of the botmaster and bots trying to communicate with other by increasing the hacker count in the GUI and also by logging the details of the host and destination machines in a specified format for later analysis and mitigation. With inbuilt features like connection timeout and waiting period in the low interaction honeypot, it is possible to break connections within a stipulated time period and also prevent dos attacks.

In the server side of the botnet architecture, various attributes are added to the database botmaster's table named bots with the help of which necessary information can be retrieved for further analysis and observation. In order to minimise the risk of any massive attack on the server system, timeout period and waiting period are specified. The impact of the attack is defined from the impact of the malware sent by the attacker. Transfer of any such malware packets would get detected in the honeypot residing in the bot machine and then instantaneously report the botmaster in the botnet and it will break the connection so that further attack can be restricted. This way the client machine also gathers the information of the IP address of the attacker. The IP address information can further be used to track the actual botmaster i.e the origin of the attack can be detected and mitigation techniques can be applied.

The low interaction honeypot which was chosen for its easy deployment , maintainability and low risk entailment was hence successfully deployed in all of the bot/client machine . The honeypot is kept to run continuously in the background waiting for client connection via FTP protocol at port no 21. On connection establishment between bots and the botmaster the honeypot was successful in its detection and logging informations.

6.2 Future Scope

- Botnet need not always be considered only as threat in the security community. The phenomenon can also be used in ethical sense by security professionals when it is used as a trap for defence against the threat. A botnet architecture like the one implemented can be created with a facade botmaster and bots. As it is disguised from the real attackers, whenever a real botmaster tries to harm the bots by performing any malicious activities , the facade system created can monitor the activities and report abuse or simply observe the activities of the real attackers in the network for later analysis. This method can especially be useful when both anomaly detection technique and mining based techniques for passive analysis are merged to form a hybrid architecture for efficient detection and mitigation schemes.
- After detection, efficient mitigation techniques are also left as further scope for this thesis. Mitigation techniques might include blockage of the IP address of the detected attacker from the network and intimating all those concerned for non compliance with such malicious systems.
- Instead of using a single honeypot, we can use a network of honeypots to deploy in both the bot and botmaster systems. Use of honeynets will increase the level of security which will leave the attackers to fetch more time for accessing several honeypots instead of one. Connecting honeypots outside of the university network to lure attackers especially disguised as attractive e-commerce sites etc can attract and trap more real attackers.
- The connection time when the bots and the botmaster communicates with each other can be increased so that a more detailed analysis can be done and all information sent from the botmaster can be attained.
- Current work focuses only on the FTP protocol as the main communication module although TCP is the basic medium of communication used. Others protocols such as commonly used IRC channels, HTTP etc can be implemented in the same architecture and the honeypot can also be built to run on those protocols. The port used for listening to connections can be changed from the standard FTP port 21 to any user given port number from the wide range of allowed ports.
- As an interesting scope, the whole botnet architecture and the detection system can be deployed in a cloud infrastructure. However a more scalable and robust architecture

would be required for the same. This might involve development of a good virtual honeypot or virtual honeynets to be deployed efficiently on a cloud domain.

- The centralized botnet architecture can be extended to a P2P domain or hybrid network architecture depending on the latest trends of attackers. With the increasing popularity of mobile botnets, a P2P architecture must be made more robust and efficient.
- The current architecture could also be tried to deploy honeypots other than low interaction ones. That would include deployment of medium and high interaction honeypots and comparison made. A high interaction honeypot system especially would emulate a full operating system or host complete services. A DDoS attack can be attracted by use of such high interaction honeypots and hence an efficient trapping mechanism might be built.

Chapter 7 REFERENCES

- [1] H. R. Zeidanloo, M. S. M. Zamani, M. J. Z. shooshtari and P. V. Amoli, "A Taxonomy of Botnet Detection Techniques," *IEEE*, 2010.
- [2] Z. Bu, R. Kashyap and A. Wosotowsky, "The New Era of Botnets," McAfee Labs, 2010.
- [3] M. Eslahi, R. Salleh and N. B. Anuar, "MoBots: A New Generation of Botnets on Mobile," in *ISCAIE 2012*, Kinabalu Malaysia, 2012.
- [4] P. Diebold, A. Hess and G. Sch afer, "A Honeypot Architecture for Detecting and Analyzing Unknown Network Attacks," in *KiVS05*, Germany, 2005.
- [5] K. L. Kyaw and P. Gyi, " Hybrid Honeypot System for Network Security," *World Academy of Science, Engineering and Technology*, Vols. Vol:2 2008-12-29 , 2008.
- [6] A. KARIM, R. B. SALLEH, M. SHIRAZ and e. al, "Botnet detection techniques: review, future trends, and issues," *Botnet detection techniques: review, future trends, and issues*, 2014.
- [7] N. Raghava, D. Sahgal and S. Chandna, "Classification of Botnet Detection Based on Botnet Architecture," in *Internationsl Conference on Communication systems and Network Technologies*, 2012.
- [8] P. Wang, S. Sparks and C. C. Zou, "An Advanced Hybrid Peer-to-Peer Botnet," in *USENIX*, 2007.
- [9] G. Fedynyshyn, M. C. Chuah and G. Tan, "Detection and Classification of Different Botnet C&C Channels," in *ATC*, 2011.
- [10] M. Feily, A. Shahrestani and S. Ramadass, "A Survey of Botnet and Botnet Detection," in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, 2009.
- [11] J. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *Proceedings of USENIX*, 2006.
- [12] S. Siboni and A. Cohen, "Botnet identification via universal anomaly detection," in *Information Forensics and Security (WIFS), IEEE International Workshop*, 2014.

- [13] G. Gu, J. Zhang and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels".
- [14] G. Gu, R. Perdisci, J. Zhang and W. Lee, "BotMiner:clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *SS'08 Proceedings of the 17th conference on Security symposium*, Berkeley, CA, USA ©2008 , 2008.
- [15] R. A. G. Rodriguez and M. Fernandez, "ANALYSIS OF BOTNETS THROUGH LIFE-CYCLE," in *SECRYPT 2011 - International Conference on Security and Cryptography*, 2011.
- [16] N. Hachem, Y. B. Mustapha, G. G. Granadillo and H. Debar, "Botnets: Lifecycle and Taxonomy," in *IEEE*, Evry, France, 2011.
- [17] P. Bacher, T. Holz, M. Kotter and G. Wicherski, "Know your Enemy: Tracking Botnets Using honeynets to learn more about Bots".
- [18] N. Provos, "A Virtual Honeypot Framework," in *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, 2004.
- [19] J. Bhatia, R. Sehgal and S. Kumar, "Botnet Command Detection using Virtual," *International Journal of Network Security & Its Applications*, vol. Vol.3, no. 5, 2011.
- [20] C. O. Varian, R. Rughini and O. Purdila, "A Practical Analysis of Virtual Honeypot Mechanisms," in *9th RoEduNet IEEE International Conference* , 2010.
- [21] E. Cooke, F. Jahanian and D. McPherson, "The Zombie Roundup:Understanding, Detecting, and Disrupting Botnets," in *SRUTI'05 ,USENIX Association*, Berkeley, CA, USA, 2005.
- [22] F. C. Freiling, T. Holz and G. Wicherski, in *Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks*, Springer, pp. 319-335.
- [23] H. Parimala and K. Balamurugan, "ACHIEVING HIGHER NETWORK SECURITY BY PREVENTING DDOS ATTACKS USING HONEYPOT," *International Journal of Computer Network and Security (IJCNS)*, vol. Vol 6.1, 2014.
- [24] J. Francois, I. Aib and R. Bouta, "FireCol: A Collaborative Protection Network For the Detection of Flooding DDoS Attacks," in *IEEE*, 2012.
- [25] G. Gu, P. Porras, V. Yegneswaran and M. Fong, "BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation," in *16th USENIX Security Symposium*.
- [26] A. Karasaridis, B. Rexroad and D. Hoeflin, "Wide-Scale Botnet detection and

- characterization," *ACM DL*, 2007.
- [27] A. Brodsky and D. Brodsky, "A Distributed Content Independent Method for Spam Detection," in *HotBots'07 Proceedings*, 2007.
- [28] Y. Xie, F. Yu, K. Achan, R. Panigrahy and G. Hult, "Spamming Botnets: Signatures and characteristics.," in *ACM SIGCOMM'08*, Seattle, 2008..
- [29] I. Mokube and M. Adams, "Honeypots: Concepts, Approaches, and Challenges," in *ACM-SE 45 Proceedings of the 45th annual southeast regional conference*, 2007.
- [30] J. G. J. B. B. GRIZZARD and H. L. Owen, "Using Honeynets to Protect Large Enterprise Networks," *THE IEEE COMPUTER SOCIETY*, 2004.
- [31] L. Yongli, Z. Jie, W. Shufang and W. Zixian, "Model and Evaluation of a New Honeynet," in *2012 IEEE Symposium on Robotics and Applications(ISRA)*, 2012.
- [32] L. SPITZNER, "The Honeynet Project:Trapping the Hackers," *IEEE SECURITY & PRIVACY*, MARCH/APRI 2003.
- [33] B. MCCARTY, "Botnets: Big and Bigger," *IEEE SECURITY & PRIVACY*, JULY/AUGUST 2003.
- [34] "Know Your Enemy:GenII Honeynets," 12 May 2005. [Online]. Available: <http://old.honeynet.org/papers/gen2/index.html>. [Accessed 29 June 2015].
- [35] A. S., C. A, M. G and Z. J, "A Technique for Detecting New Attacks in Low-Interaction Honeypot Traffic," in *Internet Monitoring and Protection, 2009. ICIMP '09. Fourth International Conference*, Venice/Mestre, 2009.
- [36] A. Ramani, S. Chamotra, J. Bhatia and R. Kamal, "Deployment of a low interaction honeypot in an organizational private network," in *ETNCC,IEEE*, Udaipur, 2011.
- [37] R. J. d. S. V. I, S. Andre and J. Kleinschmidt, "Capture and Analysis of Malicious Traffic in VoIP Environments Using a Low Interaction Honeypot," in *IEEE*, 2015.
- [38] P. Baecher, M. Koetter, T. Holz, M. Dornseif and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," in *RAID 2006 Springer*, Verlag Berlin Heidelberg, 2006.
- [39] "COLLECTING MALWARE FROM DISTRIBUTED HONEYPOTS - HONEYPHARM," in *IEEE GCC Conference and Exhibition (GCC, Dubai, United Arab Emirates*, 2011.
- [40] S. Kumar, R. Sehgal and P. Singh, "Nepenthes Honeypotsbased Botnet Detection,"

Journal of Advances in Information Technology, vol. 3, no. 4, pp. 215-221, 2012.

- [41] J. Ma, K. Chai, Y. Xiao and T. Lan, "High-Interaction Honeypot System for SQL Injection Analysis," in *IEEE*, Nanjing, Jiangsu, 2011.
- [42] H. Artail, H. Safa, M. Sraj, I. Kuwatly and Z. Al-Masri, "A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks," *ScienceDirect(Computers & Security)*, vol. 25, no. 4, p. 274–288, 2006.
- [43] I. Koniaris, G. Papadimitriou, P. Nicopolitidis and M. Obaidat, "Honeypots Deployment for the Analysis and Visualization of Malware Activity and Malicious Connections," in *IEEE ICC 2014 - Communications Software, Services and Multimedia Applications Symposium*, 2014.
- [44] P. Baecher, M. Koetter, T. Holz, M. Dornseif and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," *Recent Advances in Intrusion Detection (RAID)*, p. 165 – 184, 2006.
- [45] "Dionaea honeypot," [Online]. Available: <http://dionaea.carnivore.it/>.
- [46] M. Feily, S. A and R. S, "A Survey of Botnet and Botnet Detection," in *IEEE*, 2009.
- [47] W. Strayer, W. R and C. Livadas, "Detecting botnets with tight command and control," in *IEEE*, 2006.
- [48] F. Begin, "BYOB: Build Your Own Botnet," in *GIAC (GSEC)*, 2011.
- [49] "Analysis of a Botnet Takeover," in *IEEE COMPUTER AND RELIABILITY SOCIETIES*, 2011.
- [50] P. Bächer, T. Holz, M. Kötter and G. Wicherski, "Know your Enemy: Tracking Botnets".