GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Reverse-Engineering Malware: Malware Analysis Tools and Techniques (Forens
at http://www.giac.org/registration/grem

GIAC Reverse Engineering Malware (GREM)
Practical Assignment
Version 1.1 (added July 23, 2004)

Julia Hopkins

14/12/2004

Reverse Engineering "msrll.exe"

This paper is a description of the steps I took to reverse engineer a file called msrll.exe. It begins with a description my laboratory setup. Then I give an account of my behavioural and code analysis. Finally I discuss the implications of the functionality of the malware and suggest ways of avoiding infection.

© SANS Institute 2005, Author retains full rights.

CONTENTS

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

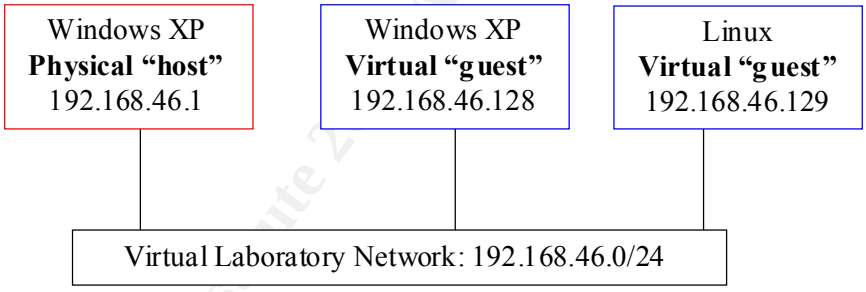
Laboratory Setup

My laboratory setup consist s of my host operating system and two virtual machines . My phys ical host is a Sony Vaio Notebook, model SVGN -S1HP. It is never connected to a production network. My host operating system is Microsoft Windows XP version 5.1.2600.1240 (Service Pack 2). On the host operating system I have installed VMWare Works tation for Windows ver sion 4.5.2 build 88:48. I have used this software to create my two virtual machines. The details are as follows: -

Virtual Machine 1: Microsoft Windows XP version 5.1.2600.1240 (Service Pack 2) , 224MB RAM, max of 4GB HD (IDE), host-only NIC

Virtual Machine 2: Linux Red Hat 9 – the Virtual Machine provided on CDROM by Lenny Zeltser during the course. "This Linux VMware machine was installed using CDs that were created from Red Hat Linux 9 ISO images downloaded from the http://www.redhat.com. Th is is a "minimal" installation that includes additional packages useful for malware analysis." [1] The virtual machine has 64 MB RAM, a max of 2GB HD, and a host-only NIC.

The diagram below shows the IP addresses of the host, the virtual network and the two virtual 'guest' machines.



© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Properties of the Malware Specimen

The malware specimen is provided as a .zip file (msrll.zip). When unzipped with Winzip it has the following properties: -

Name: msrll.exe
Type of file: msrll.exe is a Microsoft Windows executable file which has been packed with ASPack. I obtained this information by running Bintext on msrll.exe (see screen shots below). Bintext is a tool which extracts embedded strings from executables.

The string "!This program cannot be run in DOS mode" is commonly found in the first sector of windows executables and the string ".aspack" indicates that msrll.exe is packed with ASPack. The Windows dll names and functions visible in the second screen shot confirm that msr ll.exe is a Windows executable.

Size: 41,984 KB
MD5 Hash: 84acfe96a98590813413122c12c11aaa
Operating system it runs on: Windows XP and other MS Operating systems
Strings embedded into it: Most of the strings found by Bintext are obfuscated because the malware is packed. However, the following screenshots show some strings that are not obfuscated: -

File pos	Mem pos	ID	Text
A 0000004D	0040004D	0	!This program cannot be run in DOS mode.
A 00000178	00400178	0	.text
A 000001A0	004001A0	0	.data
A 000001F0	004001F0	0	.idata
A 00000218	00400218	0	.aspack
A 00000240	00400240	0	.adata

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Behavioural Analysis

I started my analysis of the malware specimen using behavioural analysis. When I could go no further down a particular path of observation using this kind of analysis I returned to the path later using code analysis. The following is a description of my behavioural analysis:-

The first stage of my behavioural analysis involved the use of five system monitoring tools to find out what would happen in the background if I ran the malicious executable on my Windows XP virtual machine. I also wanted to record any changes that the running malicious executable made to the system:-

1. **Regshot v1.61e5 Final -** for changes to the Registry and file system
2. **Filemon v 6.07 –** for monitoring file access
3. **Regmon v 6.06 –** for monitoring registry access
4. **TDIMon v 1.0 –** for monitoring network activity
5. **Microsoft Windows Task Manager version 5.1** – to view running processes

I began by using MS Windows Task Manager to familiarise myself with the processes running on my Windows XP virtual machine.

Then I prepared Filemon, Regmon and TDImon for action on the Windows virtual machine.

I took my first snapshot of the registry and the file system (C:\ and subdirectories) using Regshot.

Then I started the capture function on each of the three monitoring tools and quickly double clicked on the msrll.exe icon on my desktop to execute it.

I let it run for about forty seconds during which I noticed that a process called msrll.exe appeared in Task Manager.

After the forty seconds I terminated the msrll.exe process from Task Manager.

Then I paused the capture function on the three monitoring tools before taking my second snapshot with Regshot.

This is what I found:-

First of all, Task Manager showed that the malicious executable started a new process with the same name as itself, msrll.exe.

I pressed the "compare" button in Regshot to compare the two snapshots taken in the steps above. The output (see Appendix A) showed several very interesting changes:-

A new key was created:
HKLM/SYSTEM/ControlSet 001/Services/mfm/Security

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

All changes to the HKLM/SYSTEM/ControlSet001 key also appeared under the HKLM/SYSTEM/CurrentControlSet key. HKLM/SYSTEM/CurrentCont rolSet is a pointer to whichever control set was used to boot the computer which was ControSet001 in this case.

In this key, a new service was added: -

HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\Security**Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00 00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 01 00 00 00**
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm**Type: 0x00000120**
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm**Start: 0x00000002**
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm**ErrorControl: 0x00000002**
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm**ImagePath: "C:\WINDOWS\system32\mfm\msrll.exe"**
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm**DisplayName: "Rll enhanced d rive"**
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm**ObjectName: "LocalSystem"**

The display name of the service is "Rll enhanced drive" but the underlying executable (ImagePath) is C:\WINDOWS\system32\mfm\msrll.exe – the malicious executable. The Start value is set to 2 which means that the service starts automatically when the operating system is started. The ObjectName value is set to LocalSytem which means that the service (msrll.exe) is run s with system privileges.

Interestingly, a new value wa s added to the Registry which tells Windows XP's built in firewall to enable msrll.exe for any source. In other words, the malware is configuring the firewall to let it talk out:

HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \SharedAccess \Paramet ers\FirewallPolicy\StandardProfile \AuthorizedApplications \List**C:\WINDOWS\system 32\mfm\msrll.exe: "C:\WINDOWS\system32\mfm\msrll.exe:*:Enabled:msrll"**

Regshot shows that a new folder called **mfm** was created in C: \WINDOWS \system32 and that two new files were dropped into the mfm folder. The files are called **jtram.conf** and **msrll.exe**. The malware also deleted itself from the desktop.

Incidentally, the following internet cache files were modified but I am uncertain whether this has any significance.
C:\Documents and Settings \julia\Cookies\index.dat
C:\Documents and Settings \julia\Local Settings \History\History.IE5 \index.dat
C:\Documents and Settings \julia\Local Settings \Temporary Internet Files\Content.IE5 \index.dat

Quick Summary of Regshot findings

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

The malware creates two new files in C:\windows\system32\mfm. One of the files is an exact copy of the original which I established by running MD5Sum on it to obtain its hash value. The other is called jtram.conf, perhaps a configuration file for msrll.exe. I opened it with notepad and below is a screenshot of what it contained: -

The contents of jtram.conf are a set of what look like18 encrypted values, all ending in "= =" which is the "assignment" function in C. Six of them are assigned to what I think is a NULL character, perhaps for initialisation. Fifteen of the values are 50 characters long. Two of the values are 74 characters long and one value is 126 characters long. I stopped and started msrll.exe several times and established that jtram.conf contains a completely different set of values every time. The MD5 hash of jtram.conf therefore changes each time the malware runs.

The malware also creates a new service configured to run every time the computer starts up. Its display name is "RII enhanced drive" but the underlying executable is the malware itself, msrll.exe. The service runs with system privileges. The malware also tells Windows XP Internet Connection Firewall to allow msrll.exe to "talk out". Finally, msrll.exe deletes itself from the place that it is executed from in the first place.

Regmon offered nothing new. It did however confirm that the malware creates a service called RII enhanced drive and that it is actually created by services.exe working on behalf of the malware.

Filemon showed the malware looking for various dll files in the location it was executed from (the desktop in this case). When it couldn't find them there it looked in C:\Windows\system32 for them. Filemon confirmed all the file-related activity found by Regshot. However, when these transactions were over, Filemon recorded further activity:-

The malware persistently looked for a file called rsaenh.dll, a cryptographic service provider, in the mfm folder. When it couldn't find it there it looked for it in C:\Windows\system32. It found it here and read from it. This indicates that the malware uses encryption.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Then it repeatedly attempted to open the path C:\dev\random. I knew that this directory didn't exist on my system and considered that perhaps this folder existed on the malware author's computer when he was creating the malware and he forgot to change the path before distributing his virus.

Each time the malware failed to open C:\dev\random, it opened jtram.conf and wrote something to it. Then it tried to access the folder again. Eventually it gave up and closed jtram.conf. This is obviously the point at which jtram.conf gets its contents but where are they coming from if the malware can't access C:/dev/random? Maybe they are coming from the file that the malware spent some time reading from just before this step - rsaenh.dll.

I tried creating the c:\dev\random directory on my Windows VM and running Filemon again to see what would happen. The malware behaved in exactly the same way as before even though it was able to open the C:\dev\random folder this time.

However, I ran an internet search on C: \dev\random and discovered the probable reason that the malware was trying to access this folder. It turns out that /dev/random is a feature of the Linux kernel and of certain *BSD kernels. It is a character device that provides you with "high quality, cryptographically strong, random data"[1]. This behaviour ties in with the previous observation of the malware opening rsaenh.dll, a cryptographic service provider. Maybe rsaenh.ll's encryption process requires some random input, or what is called a "salt" in encryption terms. It is unusual that the author has included this capability in his code because behaviour so far indicates that the Trojan is targeted at MS Windows operating systems. However, MS Windows operating systems don't support the dev/random device. Maybe the malware was compiled on a Linux operating system. It is good news for me if the random aspect of the author's text obscuration plans doesn't work!

TDImon revealed that msrll.exe is listening on TCP ports **2200** and **113** on all local ip addresses (0.0.0.0:2200 and 0.0.0.0:113). TCP Port 113 is used for Ident requests. When an IRC server receives a connection request it will typically send an Ident request to the connecting client on TCP port 113 to establish the connecting user's identity. This is the first indication that the malware may be trying to connect to an IRC server. In order to ensure a successful connection it needs to listen on port 113 to provide the necessary identification information to the irc server. TCP port 2200 is not commonly used for anything in particular. Maybe it is being held open as a back door to the infected computer.

TDImon also showed scvhost.exe making a DNS request from UDP port 1042 (to 192.168.46.1:53).
The relevant lines from the TDIMon output log can be found in Appendix A.

Having seen that the malware is instigating some network traffic, I decided to capture some packets using a sniffer located on my Linux virtual machine. I started Snort v2.0.4 in promiscuous mode from the Linux virtual machine and double-clicked on

[1] EGD: The Entropy Gathering Daemon by Brian Warner - http://egd.sourceforge.net

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

the instance of msrll.exe that the malware created previously in C:\Windows\system32\mfm. My findings were as follows: -

I saw DNS requests coming from the inf ected machine, confirming my TDImon findings. The infected machine was trying to resolve a host called **collective7.zxy0.com** :-

I also saw lots of network activity taking place between UDP port 137 on the infected machine and UDP port 137 on the broadc ast ip address (192.168.46.255) and also from UDP 138 on the infected machine to the broadcast address. I understand that UDP port 138 is sometimes used in Netbios exploits but UDP ports 137, 138 and 139 are used by Windows to broadcast information relatin g to shares. I decided to concentrate on the hostname that the malware is trying to resolve.

My next step was to mould my environment by allowing the malware to resolve the hostname collective7.zxy0.com. I added an entry to the hosts file on my Windows X P virtual machine (C:\Windows\system32\drivers\etc\hosts) to link collective7.zxy0.com with my Linux virtual machine (192.168.46.129). Then I ran the sniffer again.

Now that the malware was able to resolve collective7.zxy0.com, some new network activity began. First of all the malware tried to connect to TCP port **6667** on my Linux virtual machine. However, each tim e the malware sent a “Synchroniz e” packet to port 6667, it received an “Acknowledge, Reset”. The malware also tried but failed to connect to TCP ports **9999** and **8080** on the Linux virtual machine. It also received “Resets” for these connection attempts.

Msrll.exe was trying to connect to TCP port 6667 on collective7.zxy0.com which I had resolved to point to my Linux virtual machine. TCP port 6667 is typically associated with IRC servers. So again I moulded my environment by setting up an irc server (ircd) to listen on TCP port 6667 on my Linux virtual machine. I ran Snort again to capture any changes in the malware’s behaviour.

This time the malwa re’s connection attempt to TCP port 6667 was successful . After the completion of the three -way handshake, msrll.exe sent the following packet to the irc server:-

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
11/15-01:37:40.808413 192.168.46.128:1092 -> 192.168.46.129:6667
TCP TTL:128 TOS:0x0 ID:294 IpLen:20 DgmLen:131 DF
***AP*** Seq: 0x7ED8C896  Ack: 0xB600931C  Win: 0xFAF0  TcpLen: 20
55 53 45 52 20 75 4D 44 57 49 55 57 54 50 58 20  USER uMDWIUWTPX
6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A 64 6D 54  localhost 0 :dmT
52 42 4C 66 53 63 70 55 72 49 70 6F 64 6B 52 4F  RBLfScpUrIpodkRO
4C 64 44 6B 4A 75 4D 53 42 6E 68 54 47 4F 46 53  LdDkJuMSBnhTGOFS
4A 4A 47 74 55 76 73 77 57 0A 4E 49 43 4B 20 52  JJGtUvswW.NICK R
55 58 50 68 78 58 6B 71 64 7A 0A                 UXPhxXkqdz.
```

Two keywords stood out in the ascii text – USER and NICK. This is the standard way of logging on to an irc server. The malware (I may refer to the malware as a bot from now on as that is the name given to malware that uses irc.) has provided an obfuscated string of characters for its real name and another obfuscated string for its nickname. Later monitoring showed that these strings change each time the bot connects to the irc server so they must be randomly generated somehow. This goes against my assumptions concerning calls to "dev/random" not working. Maybe the malware has access to another random number generator.

The irc server replied to the above packet with the following packet: -

```
11/15-01:37:40.809603 192.168.46.129:6667 -> 192.168.46.128:1092
TCP TTL:64 TOS:0x0 ID:46074 IpLen:20 DgmLen:86 DF
***AP*** Seq: 0xB600931C  Ack: 0x7ED8C8F1  Win: 0x16D0  TcpLen: 20
4E 4F 54 49 43 45 20 41 55 54 48 20 3A 2A 2A 2A  NOTICE AUTH :***
20 4C 6F 6F 6B 69 6E 67 20 75 70 20 79 6F 75 72   Looking up your
20 68 6F 73 74 6E 61 6D 65 2E 2E 2E 0D 0A         hostname.....
```

In order to identify the host, a connection was then made from TCP port 1027 on the machine that the irc server is on to TCP port 113 on the infected machine. This was the ident request that I had forseen. The packet contained the string "1092, 6667" which as you can see from the screenshot above are the ports being used for the irc connection on the respective machines. In reply, the windows virtual machine sent the string "1092, 6667: USERID: UNIX: BwMsQZBNF. Again the text was obscured but the irc server was satisfied, confirming with another packet that it had received the ident response. Eventually, the irc server sent a long welcome message to the infected machine, starting with the following :-

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
11/15-01:38:08.552099 192.168.46.129:6667 -> 192.168.46.128:1092
TCP TTL:64 TOS:0x0 ID:46077 IpLen:20 DgmLen:1064 DF
***AP*** Seq: 0xB60093C1  Ack: 0x7ED8C8F1  Win: 0x16D0  TcpLen: 20
3A 6C 6F 63 61 6C 68 6F 73 74 2E 6C 6F 63 61 6C  :localhost.local
64 6F 6D 61 69 6E 20 30 30 31 20 52 55 58 50 68  domain 001 RUXPh
78 58 6B 71 20 3A 57 65 6C 63 6F 6D 65 20 74 6F  xXkq :Welcome to
20 74 68 65 20 49 6E 74 65 72 6E 65 74 20 52 65   the Internet Re
6C 61 79 20 4E 65 74 77 6F 72 6B 20 52 55 58 50  lay Network RUXP
68 78 58 6B 71 0D 0A 3A 6C 6F 63 61 6C 68 6F 73  hxXkq..:localhos
74 2E 6C 6F 63 61 6C 64 6F 6D 61 69 6E 20 30 30  t.localdomain 00
32 20 52 55 58 50 68 78 58 6B 71 20 3A 59 6F 75  2 RUXPhxXkq :You
72 20 68 6F 73 74 20 69 73 20 6C 6F 63 61 6C 68  r host is localh
6F 73 74 2E 6C 6F 63 61 6C 64 6F 6D 61 69 6E 5B  ost.localdomain[
6C 6F 63 61 6C 68 6F 73 74 2E 6C 6F 63 61 6C 64  localhost.locald
6F 6D 61 69 6E 2F 36 36 36 37 5D 2C 20 72 75 6E  omain/6667], run
6E 69 6E 67 20 76 65 72 73 69 6F 6E 20 32 2E 38  ning version 2.8
2F 68 79 62 72 69 64 2D 36 2E 33 2E 31 0D 0A 4E  /hybrid-6.3.1..N
4F 54 49 43 45 20 52 55 58 50 68 78 58 6B 71 20  OTICE RUXPhxXkq
3A 2A 2A 2A 20 59 6F 75 72 20 68 6F 73 74 20 69  :*** Your host i
73 20 6C 6F 63 61 6C 68 6F 73 74 2E 6C 6F 63 61  s localhost.loca
6C 64 6F 6D 61 69 6E 5B 6C 6F 63 61 6C 68 6F 73  ldomain[localhos
74 2E 6C 6F 63 61 6C 64 6F 6D 61 69 6E 2F 36 36  t.localdomain/66
36 37 5D 2C 20 72 75 6E 6E 69 6E 67 20 76 65 72  67], running ver
```

And ending with:-

```
11/15-01:38:08.921664 192.168.46.129:6667 -> 192.168.46.128:1092
TCP TTL:64 TOS:0x0 ID:46079 IpLen:20 DgmLen:116 DF
***AP*** Seq: 0xB6009954  Ack: 0x7ED8C904  Win: 0x16D0  TcpLen: 20
3A 6C 6F 63 61 6C 68 6F 73 74 2E 6C 6F 63 61 6C  :localhost.local
64 6F 6D 61 69 6E 20 33 30 32 20 52 55 58 50 68  domain 302 RUXPh
78 58 6B 71 20 3A 52 55 58 50 68 78 58 6B 71 3D  xXkq :RUXPhxXkq=
2B 42 77 4D 73 51 5A 42 4E 46 40 31 39 32 2E 31  +BwMsQZBNF@192.1
36 38 2E 34 36 2E 31 32 38 20 0D 0A              68.46.128 ..
```

The string, “RUXPhxXkq” which was the value of NICK in a previous packet appears many times in the welcome message . In the last part of the welcome message we see the string

“RUXPhxXkq=+BwMsQZBNF@192.168.46.128 ”

The irc server is associating the nickname with the USERID that it received in the ident request earlier. This is standard irc behaviour.

The next packet showed the bot joining a channel called **#mils** on the irc server (JOIN #mils :.).

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
11/15-01:38:12.919763 192.168.46.128:1092 -> 192.168.46.129:6667
TCP TTL:128 TOS:0x0 ID:305 IpLen:20 DgmLen:53 DF
***AP*** Seq: 0x7ED8C904  Ack: 0xB60099A0  Win: 0xFAA4  TcpLen: 20
4A 4F 49 4E 20 23 6D 69 6C 73 20 3A 0A           JOIN #mils :.
```

.

Unlike for the values for NICK, USER and USERID, #mils stayed the same each time the bot connected to the irc server . The stop at the end of the packet means 'new line'.

The malware then checked the mode of the channel and the nicknames of those currently present on the channel (MODE #mils.WHO #mils.) :-

```
11/16-00:24:11.546273 192.168.46.128:1057 -> 192.168.46.129:6667
TCP TTL:128 TOS:0x0 ID:256 IpLen:20 DgmLen:61 DF
***AP*** Seq: 0x1A9C622  Ack: 0xDC78FA45  Win: 0xF9CE  TcpLen: 20
4D 4F 44 45 20 23 6D 69 6C 73 0A 57 48 4F 20 23  MODE #mils.WHO #
6D 69 6C 73 0A                                   mils.
```

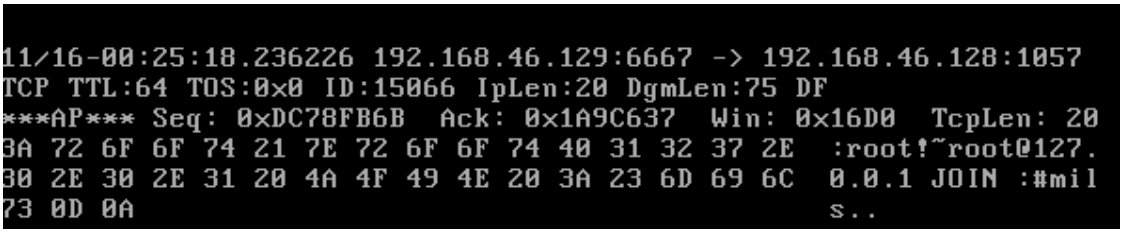
Maybe it was storing these details for the author to pick up at a later date. Or maybe it was comparing the results of these commands to predetermined criteria. It may on the other hand, be looking for a particular nickname, maybe that of the author. If so, it is possible that the nickname of the author is hard -coded into the bot's source code. The same goes for the mode of the channel. Perhaps the Trojan requires the channel to be in a particular mode for some reason. I investigate this further in code analysis.

The irc server responded to these three commands with details of the malware user only as there was no body else connected to the channel .

I assumed that the bot was connecting to the irc channel in order to receive instructions from its author. So I connect ed to the channel myself to see if I could communicate with the malware and therefore learn more about i t. I started an irc client on the Linux virtual machine and logged in. At the prompt I typed "**/JOIN #mils"** to join the channel. I also tried typing " **/JOIN #mils :**" . I thought that the bot could be using the colon character as a key for the channel (see Sn ort log screenshots above).

When I joined the #mils channel (without a key) , the snort log showed the irc server sending a notification of my appearance on the channel to the malware specim en on the infected machine. This is standard behaviour for irc se rvers. However, it may be worth intercepting this behaviour during code analysis to see if the malware does a text comparison between the name of the new channel member and the name of the author. This is something I could return to in my code analysis later:-

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.



I tried out a few irc commands to see if I could get a response from the malware. When I entered the command **/NAMES**, I found that someone called @sYAyBLAoY was also logged on to the channel. The @ sign usually means that a user has operator status on an irc channel. **/WHOIS sYAyBLAoY** got the following response: -

is DIQQNFWitA@192.168.46.128 (QaRdoJNJHZkdNLJkwR) on channels: @#mils on irc server localhost.localdomain (IRC Server) sYAyBLAoY has been i dle 30 mins

It is obvious that this user is the bot as the ip address 192.168.46.128 is that of my infected virtual machine . In this case the USERID the malware is using is DIQQNFWitA. It has be en waiting for 30mins. Perhaps it i s waiting for a command . The Snort logs showed some periodic "Ping-Pong" activity occurring over the irc connection which can be attributed to the malware trying to keep the irc client alive , again showing that it is waiting for something .

I tried a few other commands without m uch success . It was time to unpack the malware specime n and run Bintext on the unpacked version . Maybe the Bintext output would contain a list of special commands for communicating with the malware on the irc channel.

When I ran Bintext on msrll.exe at t he start of the analysis, one of the strings found was ".aspack" wh ich indicated that the malware was packed with popular packer ASPack. To save time in unpacking the executable I downloaded ASPackDie [2] from the Internet. I pointed ASPackDie to a copy of ms rll.exe on my desktop and it unpacked the malware to a file called "unpacked.exe", also on my desktop. The first thing I did was to run Bintext on it. A complete list of the embedded strings produced by Bintext can be found in Appendix B.

I scoured the list for command-like strings that I could use to try to communicate with the bot on the irc channel. There were a group of strings beginning with "?" which looked like commands such as "?login", "?status", "?kill" .

Returning to the IRC session, I tried ou t some of the commands beginning with "?" on the #mils channel. However, even though the malware instance was also present on the channel, none of the commands had any affect whatsoever. I tried typing commands with parameters such as '?login wrongpassword ' to see if I would get an error response from the Trojan but it still didn't react. I used Snort to see if any network activity occurred when I entered various commands but all I saw was the trojan sending an Ack packet back to the irc server, acknowledg ing the command but not doing anything with it.

[2] ASPackDie v1.4.1 downloaded from http://mitglied.lycos.de/yoda2k/proggies.htm

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Having failed to communicate with the bot on the irc server, I decided instead to move on to analysing the behaviour of the malware regarding its connection attempts to TCP ports 9999 and 8080. The malware tried to connect to these ports after it was able to connect to what it thought was the collective7.zxy0.com host. I wasn't sure what services the malware was looking for on these ports so I ran NetCat as a listener on each port in turn, starting with TCP port 9999. As an aside, it is worth mentioning that if the malware managed to connect to port 6667 on the collective7.zxy0.com host then it no longer attempted to connect to TCP ports 9999 and 8080.

First of all I shut down the irc server and client that I had started on my Linux virtual machine. Then I set NetCat to listen on TCP port 9999 as follows: -

```
[root@localhost root]# nc -l -p 9999 > /tmp/nc9999.log
[root@localhost root]# more /tmp/nc9999.log
USER ASisT localhost 0 :sGVRTREkqgAqnYDoFMOGJrmOcKBecWtrgwVDXJyJxLzF
NICK TyJdfPQqpN
[root@localhost root]# _
```

Netcat's output showed the data that was sent to port 9999 by the malware. It looks very much like another attempt to log on to an irc server. The USER and NICK keywords associated with irc connections are there but again their values look as if they are encrypted /random. Later tests showed that the values for USER and NICK changed every time. Maybe in the real world, there is an irc server listening on TCP port 9999 on the collective7.zxy0.com host. I don't have an irc server which listens on this port so I moved on to examine TCP port 8080 connection attempts in the same way.

```
[root@localhost root]# nc -l -p 8080 > /tmp/nc8080.log
[root@localhost root]# more /tmp/nc8080.log
USER cmKsnQSAOm localhost 0 :mgvRPeWrGiahjGPxavdERoNwFWmhPn
NICK BoDsMRNPshJg
[root@localhost root]# _
```

The results are the same as those for port 9999. I will assume that an irc server has been configured to listen on these ports on the collective7.zxy0.com.host as well as on TCP port 6667.

There is one more thing for me to look at in my behavioural analysis. One of the first things I found out about the Trojan was that it was listening on TCP ports 113 and 2200. I established that port 113 was being used to listen for Ident requests and made the assumption that TCP port 2200 was just a simple back door. I tried telneting to TCP port 2200 on the infected machine from my Linux virtual machine and was presented with the following window: -

```
[root@localhost root]# telnet
telnet> open 192.168.46.128 2200
Trying 192.168.46.128...
Connected to 192.168.46.128.
Escape character is '^]'.
#:_
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

I tried typing some simple commands but nothing worked. It is possible that this backdoor was added by the author as an afterthought and maybe even to leave a door open for other hackers to exploit.

Code Analysis

Having taken my behavioural analysis as far as I could, it was time to find out more using code analysis. I had already performed some very basic code analysis by a) running Bintext on the packed executable, b) unpacking the executable with ASPackDie and c) running Bintext on the unpacked executable. The strings that I obtained by running Bintext on the unpacked executable helped me to speculate as to what the Trojan was doing and allowed me to try out some commands on the irc channel. The full list of strings found by Bintext can be found in the Appendix B. Throughout both my behavioural and code analysis I constantly referred back to the strings that were found in the unpacked executable for ideas and clues. At the end of my code analysis I speculate on further functionality of the malware using the embedded strings as a guide.

My behavioural analysis ended at four brick walls which I list here: -

1. The Trojan connected to my irc server on port 6667 on host collective7.zxy0.com and seemed to be waiting for a command. However, I couldn't get it to respond to anything. I hoped to be able to establish with code analysis whether or not:
    a. The trojan would accept commands from anybody as long as the command was entered correctly
    b. it required certain criteria to be met in order to be able to respond to commands e.g. the MODE of the channel needed to be set to something special and/or a particular user had to be present on the channel, i.e. the author.

2. The Trojan was listening on TCP port 2200 on the infected machine but again wouldn't respond to any commands I issued to this port via a telnet session from my Linux virtual machine. Maybe my code analysis would give me an idea as to what the Trojan is expecting to receive on this port.

3. If the Trojan wasn't able to connect to port 6667 on collective7.zxy0.com it tried to connect to TCP ports 9999 and 8080. I put listeners on these ports on my Linux virtual machine and recorded the Trojan connecting to them. The Trojan seemed to be expecting irc servers to be listening on these ports too. It authenticated to them in exactly the same way as it did with my irc server on port 6667. I could configure Honeyd to emulate an irc server listening on these ports but there is little point in doing this until I can get over my first brick wall.

4. There is also the fact that most of the communications coming from the Trojan seem to be encrypted. As if that wasn't enough the communication strings also seem to be randomised. Many of the strings found by Bintext support this. A full list of relevant strings can be found in Appendix C.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

The main focus of my code analysis is therefore working out how to communicate with the Trojan.

The tools that I used were:
Bintext – to find embedded strings
IDAPro – to disassemble executable into assembly code
OllyDebug – to step through disassembled code
PEInfo – to view the malware's imported files and functions and also its structure ,
Snort – to sniff packets off network

I thought that if I could locate one of the h ard-coded irc-like commands in the code of the malware then I might be able to see how the malware expected to receive them , e.g. the format of the commands and any hard coded parameters such as passwords.

Having already unpacked the malware specime n to the desktop, the next time I double-clicked on the unpacked version, it overwrote the packed version of msrll.exe in the C:\Windows\System32\mfm folder and deleted itself from the desktop. So, I loaded the unpacked C:\Windows\system32\mfm\msrll.exe into IDAPro to disassemble it and began my analysis by searching (Alt+T) the assembly code for the '?login' string.

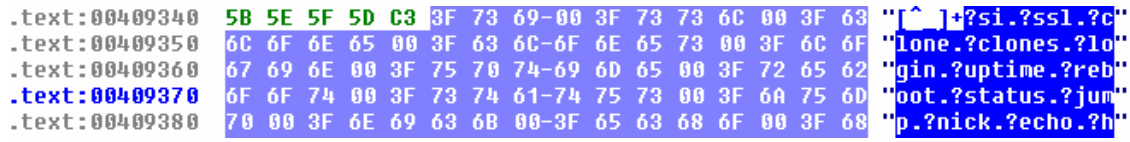
However, IDAPro could n't find "?login". I tried searching for other irc-like command strings e.g. '?status' and '?kill' but IDAPro couldn't find t hose either. This didn't make sense. I checked out the list of Strings that IDAPro automatically generated on loading msrll.exe but the irc-like command-strings weren't there either. Bintext said that the string was supposed to be at memory location 0040935D but in IDAPro the addresses jumped from 00409345 to 004094B1. However, starting from 00409345 were a large group of dd (double word) declarations in hexadecimal.

```
.text:00409345                  dd 69733Fh, 6C73733Fh, 6C633F00h, 656E6F
.text:00409345                  dd 676F6C3Fh, 3F006E69h, 69747075h, 3F00
.text:00409345                  dd 3F00746Fh, 74617473h, 3F007375h, 706D
.text:00409345                  dd 3F006B63h, 6F686365h, 75683F00h, 3F00
.text:00409345                  dd 6F6A3F00h, 3F006E69h, 3F00706Fh, 706F
.text:00409345                  dd 3F006B63h, 74726170h, 75643F00h, 3F00
.text:00409345                  dd 6569643Fh, 646D3F00h, 3F007035h, 6565
.text:00409345                  dd 753F0077h, 74616470h, 683F0065h, 6E74
.text:00409345                  dd 6669663Fh, 66213F00h, 3F006669h, 6C65
```

I thought that maybe the hexadecimal equivalent of "?login" could be somewhere amongst the declarations. Maybe the declarations themselves were referenced in the code rather than their memory location. On converting a few of the declarations to ascii I found that they were indeed a list of commands but it would take too long to convert it all in order to find my ?login command. .

I switched to Hex View in IDAPro to see if I could shed any light on which of the dd declarations contained the "?login" command (see screen shot below) .

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.



On the right hand side are the ascii strings includin g "?login" and in the middle is the equivalent hexadecimal. The hexadecimal equivalent of "?login" is "3F 6C 6F 67 69 6E" but it is split over two memory addresses, 00409350 and 00409360. Therefore I would expect one of the dd declarations to contain "3F 6 C 6F" and another to contain "67 68 6E".

3F	6C	6F		67	69	6E
?	L	o		G	I	N

However, in double word declarations the hexadecimal pairs are reversed as follows:-.

6F O	6C L	3F ?		6E N	69 i	67 G

Referring back to the dd declarations, one of the de clarations contains the first three pairs:-
676F6C3Fh with and extra pair, 67 ("g") at the start and the next dd declaration, 3F006E69h contains the pairs 6E and 69 with two extra pairs at the start . If I convert these declarations back into hex I get "?l ogin.?". The final character is part of the next command.

I searched IDAPro to see if it could find any references to the first dd declaration 676F6C3Fh . No references were found for either of the dd declarations. A quick browse through some of the cod e in IDAPro showed that dd declarations are often referenced using 'dword_memory location of declaration is at'. I tried searching for dword_409350 and also for dword_409345 but IDAPro found nothing.

I was unsuccessful in using IDAPro to locate calls to the "?login" command.

During my behavioural analysis I suggested that the Trojan may be testing to see if certain criteria are met when it issues the MODE #mils and WHO #mils commands after joining the channel. Related strings found by Bintext were " **WHO %s**" at address 00403778 , "**%s logged in**" at address 00405B88, " **MODE %s -o+b %s *@%s**" at address 00404711, " **MODE %s -bo %s %s**" at address 004047E7, " **MODE %s %s**" at address 0040505A, " **mode %s +o %s** " at address 004055A3 and at address 004134B0, " **mode %s +b %s %s**" at address 004055B8 and "_setmode" at address 0051BCFA.

The "MODE" command is referred to several times in the code and in particular with the attributes 'o' and 'b' which stand for 'operator status' and 'ban' respectively in irc - speak. One the strings above, " **mode %s +o %s** ", could be used to give 'operator status' to someone. An operator has special privileges on an irc channel. You can

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

only become an operator either by being the first person to join a channel or by being given operator status by an other operator. Perhaps the Trojan gives operator status to its master when the master joins. Or maybe it gives operator status to other instances of the malware that join the channel. I should let the malware join the #mils channel before I do in further analysis to make sure that it gets operator status.

Why would the Trojan want to ban or remove a ban from another channel member? Maybe this will become clear on analysing the code.

Starting with the "WHO %s" string, Bintext said that it is locate d at 00403778.

However, in IDAPro the addresses jump ed from 00403775 to 00403781. In between were three dd (double word) declarations and a cross reference to a subroutine (sub_403783+4C).

```
.text:00403774 ; ---------------------------------------------------------------------------
.text:00403775 dword_403775    dd 25207325h, 48570A73h, 7325204Fh
.text:00403775                                         ; DATA XREF: sub_403783+4C↓o
.text:00403781 ; ---------------------------------------------------------------------------
```

This is the same issue that I came across previously when searching for the "?login" string.

The three dd declaration s were the hexadecimal values 25207325h, 48570A73h and 7325204Fh.

Hex	25	20	73	25		48	57	0A	73		73	25	20	4F
Asc	%	Space	S	%		H	W	Line Feed	S		S	%	Space	O

I changed the order of the hex values so that the first word of each 4-word value is at the end of the 4-word value and the last word is at the start of the 4-word value.

Hex	25	73	20	25		73	0A	57	48		47	20	25	73
Asc	%	S	Space	%		s	Line Feed	**W**	**H**		**O**	Space	**%**	**s**

The "WHO %s" string is there . I have highlighted it in bold.

I examined the cross-referenced subroutine sub_403783+4C mentioned above and found a reference to the memory location of the WHO %s string, dword_403775 (00403775 is the start of the dd declarations) :-

```
.text:004037C3                 add     eax, 4B0h
.text:004037C8                 push    eax
.text:004037C9                 push    eax
.text:004037CA                 push    offset aMode    ; "MODE"
.text:004037CF                 push    offset dword_403775
.text:004037D4                 push    [ebp+arg_4]
.text:004037D7                 push    0
.text:004037D9                 push    0
.text:004037DB                 call    sub_404481
.text:004037E0                 add     esp, 14h
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Not surprisingly, the string “MODE” is in the instruction above it. Looking at the above code, it seems that both “MODE”, “WHO %s” and a few other values including the value in the EAX register and the contents of what’s at memory address EBP + FF (arg_4) are passed in to a subroutine at memory location 404481 (call sub_404481) . I examined subroutine 404481 in ADIPro but could not really tell what was happening . I then viewed the code with a debugger, Ollydbg in the hope of seeing some hard - coded ascii values which the trojan may have been using for string comparisons . I opened msrll.exe in OllyDbg and set breakpoints (F2) at the following positions: - 004037DB (the call to the subroutine), 004037E0 (the instruction after the subroutine) and 00404481 (the first instructio n of the subroutine). The n I ran the malware (F9). The malware stopped at the first breakpoint. The contents of memory address EBP + arg_4 turned out to be the user id!nick@host combo for the Trojan that had joined the irc channel with a colon in front and the string “JOIN #mils” at the end like this: -

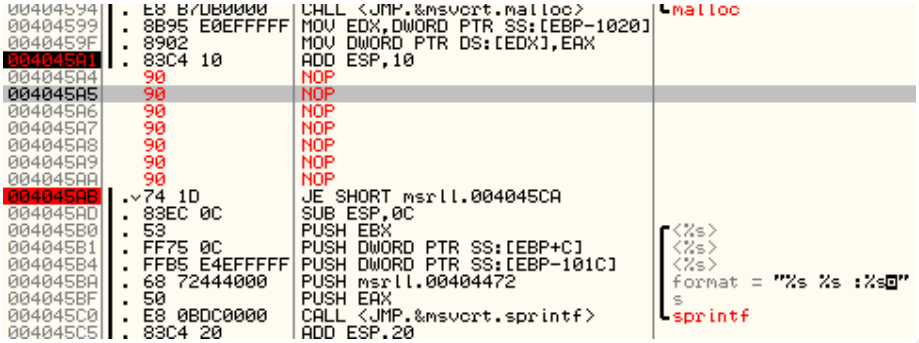
":DwCyuDrDY!cZKN@192.168.46.128 JOIN :#mils"

I pressed F7 to step into the subroutine. I stepped through each line of the subroutine (F8). The code calls subroutine 411D10 and continues. Then at instruction 411D3B the code jumps to the address that is in the EAX register, which is 404491. Then at 4044C1 the code jumps a few instructions to address 4044DC (referred to as msrll.004044DC which means that this code is part of the malware’s code rather than part of a packaged dll file or some other program). At instruction 4044E2, the magic address BAADFOOD is initialised with 0. The next jump is to 00404565. At address 404578, “USERHOST nQwRsDAiE” is printed to the screen of the irc server using the function vsnp rintf from the msvcrt dll. A comparison is made at address 4045A4 and the next line of code jumps a few lines to 4045CA if the two values that were compared were equal (JE SHORT msrll.004045CA). Each time I run the malware, this jump is never taken because the comparison is successful. However, I can see in Ollydbg that the lines of code being jumped are printing something to the screen:

```
004045A1  . 83C4 10       ADD ESP,10
004045A4  . 83BD E4EFFFFF CMP DWORD PTR SS:[EBP-101C],0
004045AB  .v74 1D         JE SHORT msrll.004045CA
004045AD  . 83EC 0C       SUB ESP,0C
004045B0  . 53            PUSH EBX                          [<%s>
004045B1  . FF75 0C       PUSH DWORD PTR SS:[EBP+C]         <%s>
004045B4  . FFB5 E4EFFFFF PUSH DWORD PTR SS:[EBP-101C]      <%s>
004045BA  . 68 72444000   PUSH msrll.00404472               format = "%s %s :%s"
004045BF  . 50            PUSH EAX                          s
004045C0  . E8 0BDC0000   CALL <JMP.&msvcrt.sprintf>        sprintf
004045C5  . 83C4 20       ADD ESP,20
004045C8  .vEB 1A         JMP SHORT msrll.004045E4
004045CA  > 83EC 08       SUB ESP,8
```

I used OllyDbg to replace the compare function at 004045A4 with a NOP instruction to stop the jump from taking place . This is called ‘patching’ and is done by selecting the line of code and then pressing the space bar. Enter ‘NOP’ in the box, hit Assemble and you will get the following. The change is only made to memory, not to disk.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.



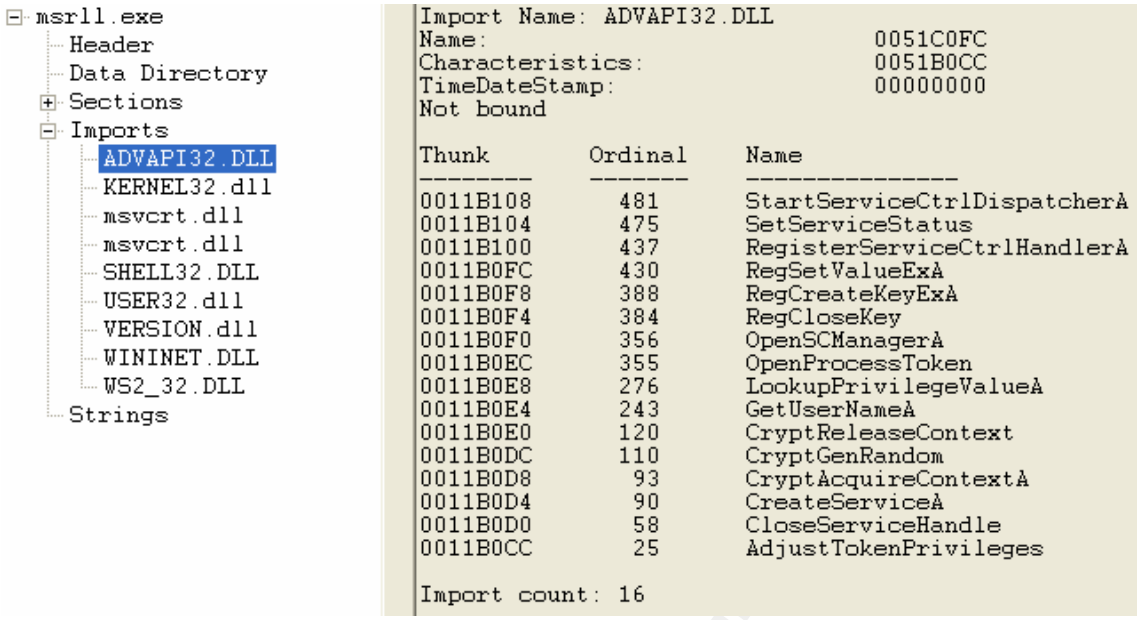
Actually, the code turned out to be much less interesting than I thought. The malware was just preparing the string USERHOST userid for printing to the irc server. Moving on, the next thing of any significance that I saw was at instruction 00403845. The userid part of the above string ("DwCyuDrDY") was being copied into the address 003D7300, the user!nick@host part of the string was being copied into 003D736D, the ip address (192.168.46.128) was being copied into 3D7C27 and "#mils" was being copied into 3D78A8. However, later debugs showed that these memory addresses changed with every run of the malware.

I gathered from the code that I had been following so far in OllyDbg that the malware was simply preparing strings about itself for introducing itself to the irc server. Nothing very exciting.

The furthest I got in following this path was to see the malware comparing the string "001 " with the following strings: "JOIN", "QUIT", "352", "302", "303", "005", "NICK", "PART", "KICK", "353", "MODE", "433" and "324". It seemed to find a match with "324" and exited the loop which was at address 00404233. This is something new. The numbers seem to represent commands. What command could the string "001" represent? Unfortunately, I was unable to establish using code analysis whether the malware was checking to see if certain conditions were met after it joined the channel.

Many of the strings found by Bintext were functions from various dlls that were imported by the malware from the infected system. I used a tool called PEInfo to see which dll files and which functions the malware was importing. Below you can see a list of dll files that the trojan requires under 'Imports' on the left and on the right, a list of functions that the selected dll file contains (I have selected ADVAPI32.dll in this example) :-

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.



Immediately, I recognise d some of the function names from my Bintext output, including CryptGenRandom and CryptAcquireContextA. I believe that the malware has even packaged up at least one whole program with itself. For example, i t is unlikely that the malware would be able to find the LibTomCrypt program on an infected computer so it seems to have packaged the whole program up with itself. I went to the features page of LibTomCrypt on the WWW (http://libtomcrypt.org/features.html) and practically every string from this page was found by Bintext. What are the bets that the features page is the same as the readme file that comes packaged with LibTomCrypt? It would be nice to be ab le to eliminate these strings from the Bintext output to make analysis simpler.

Analysis Wrap -up

The capabilities of this specimen of malware are far -reaching. Infection takes place through execution of the specimen on a MS Windows operating system an d it runs with the privileges of the logged -in user which is SYSTEM in the case of most home users. On infection it deletes itself from the place it was executed, it adjusts the Registry so that it runs as a process each time the computer is started up and it copies itself along with another file into a folder in the system path. It takes measures to hide itself from the user such as using a fake display name in Services.

The specimen then opens a backdoor on TCP port 2200. Meanwhile it tries to connect to a host called collective7.zxy0.com. If successful it tries to connect to an irc server listening on port 6667 on that host. If it fails then it tries to connect to back -up irc servers listening on ports 9999 and 8080. After connecting to port 6667 the malware joins a channel called #mils and queries the irc server to find out the mode of the channel and the names of other channel members. Then it waits. I believe that it is waiting for commands from its master.

The malware has advanced encryption capa bilities, making use of various cryptographic providers and random number generators. The userid and nickname it

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

provides to the irc server are randomly generated. Also, the malware writes several long encrypted values to the jtram.conf file that it copied into the system path. Many of the embedded strings produced by Bintext look as if they are encrypted as well as function names and file names that indicate that encryption is being used (see Appendix C for encryption/randomisation -related strings).

Even though I failed to get the bot to react to any of my commands, the embedded strings gave many clues as to what an attacker could use it for: -

Strings such as 'jolt', 'syn' and 'smurf' (see Appendix C, Attack-related strings) indicate that the master could command the bot (and perhaps hundreds of other bots at the same time) to launch various DoS attacks the victim machine or on a target other than the victim machine.

Commands such as '?reboot', '?crash' and '?rmdir' show that the bot can be used by an attacker to cause trouble on the infected machine.

A selection of strings beginning with SSL e.g. SSL_connect show that the malware is capable of talking over the WWW. Maybe it connects to a web site or web server to receive instructions from its master or maybe it can download files from a web server. Strings such as "urlopen failed" and "inetopen failed" confirm that the malware can communicate via the WWW.

There are many strings which contain 'ddc' which is a Direct -cient-to-client function in IRC (see Appendix C - DCC and IRC Socket related strings). DCC is commonly used to distribute malicious code to unsuspecting users. Perhaps the malware is using it in this way too. It could use DCC to receive updates from its master or to send data to its master.

The malware has all the capabilities needed to be a serious attack tool. It can be modified and improved upon by its attacker, making it more dangerous and versatile. It already has several built-in back-up components such as the connection attempts to ports 9999 and 8080.

The Trojan is very difficult to authenticate to – I assume that a degree of cryptanalysis would be necessary to establish the key required to be able to communicate with the Trojan. That means that it has not been designed for use by anyone who just happens upon it.

Depending on the infection rate, the #mils channel could be a meeting point for thousands of bots identical to this one. The attacker would then have the capability to launch a serious DoS attack on his adversary or on a commercial company or website or on a government website and so on. This kind of malware is being used more and more to conduct crimes of extortion. Basically, a criminal commands his zombies (the thousands of bots that have gathered in his irc channel) to launch a particular denial of service attack (e.g. jolt) against a company and refuses to stop it until a large sum of money is transferred to an off-shore bank account. It is almost impossible to track this type of attack back to its source because the DoS attacks are coming from unsuspecting users' infected machines located all over the world.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

The string "% removed" was found by Bintext. This indicates that I may be able to issue a command to the Trojan on the irc channel telling it to remove itself from the machine it has infected. However, as I can't authenticate to the Trojan, I would take the following three steps to remove it from my system: -

1. Stop the msrll.exe process using Task Manager
2. Open up Services from Control Panel and locate the servic e called "Rll enhanced drive". Right click on the service and choose properties. Change the Start-up type of the service from "Automatic" to "Disabled". This is one step towards stopping the malware from starting up automatically next time you start up your computer. However, you also need to reverse the changes that were made by the malware to your registry settings. Locate the following keys in the Registry (use Regedit from a command prompt) and delete the whole of the 'mfm' key:-

HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm

3. Delete the folder called 'mfm' from "C:/Windows/system32/".

The main piece of advice that I would give to avoid an attack by this specimen would be not to log on to your computer as System Administrator by default. This malware specimen assumes the privileges of the logged in user. Only the users with System privileges are able to make changes to the Registry. Therefore, if you were infected by the Trojan whilst lo gged in as a standard user, the malware would be unable to make the necessary changes to the Registry. That means that the service (Rll Enhanced Drive) that causes the malware to run automatically each time the computer is started up, is never created.

Secondly, monitor your computer's listening ports . Close any ports that are open and listening unnecessarily. I did not find out exactly what TCP port 2200 is used for in this particular case. However, closing it can only protect you further as this port is not commonly used for anything else.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

References

Warner, Brian. " EGD: The Entropy Gathering Daemon ". SourceForge. Jul 25 2002. Dec 27 2004.<http://egd.sourceforge.net >

Denis, Tom St. LibTomCrypt home page. 27 Dec 2004.< http://libtomcrypt.org >

Department of Physics, University of Hamburg web page . "Jolt". Physnet Security Tutorials. 27 Dec 2004. <http://www.physnet.uni-hamburg.de/physnet/security/vulnerability/jolt.html >

Huegen, Craig A. "The latest in denial of service attacks: "smurfing" description and information to minimize effects ". Feb 8 2000. 27 Dec 2004. <http://www.pentics.net/denial -of-service/white-papers/smurf.cgi>

Hyde, Randall. Art of Assembly. Webster homepage. 2000. 27 Dec 2004. <http://webster.cs.ucr.edu >

Lay Networks web page. 2000. Computer Science Tutorials . 27 Dec 2004. <http://www.laynetworks.com/assembly%20tutorials4.htm >

Tools

ASPackDie v1.4.1

Danilo, B. Sistemo website. 2003. Pro ggies web page. <http://mitglied.lycos.de/yoda2k/proggies.htm >

Snort for Linux v2.0.4 , Bintext 3.0, IDAPro v4.6 Evaluation copy , Md5sum, PEInfo, NetCat 1.10 for Unix, Ollydbg 1.10 for Windows , Regshot 1.61e5 Final for Windows , SysinternalsTDIMon v1.01 fo r Windows NT/2000/XP , Sysinternals Regmon v6.06 for Windows NT/2000/XP , Sysinternals Filemon v6.06 for Windows NT/2000/XP , Winzip 9.0 evaluation copy, Red Hat Linux 9

Zeltser, Lenny. GREM. Reverse -Engineering Malware Tools and Techniques Hands-On CDROM v9. Sans Press. 2004

VMWare Workstation 4.5.2 -8848 for Windows. < http://www.vmware.com/ >

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Appendix A

REGSHOT LOG 1.61e5
Comments:
Datetime:2004/11/15 16:18:52 , 2004/11/15 16:20:46
Computer:XPSP2 , XPSP2
Username: ,

Keys added:4

HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\Security
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet \Services \mfm
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services \mfm\Security

---
Values added:21

HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \SharedAccess \Paramet
ers\FirewallPolicy \StandardProfile \AuthorizedApplications \List\C:\WINDOWS \system
32\mfm\msrll.exe: "C: \WINDOWS \system32 \mfm \msrll.exe:*:Enabled:msrll"
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\Security\Security:
01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00 00
00 02 80 14 00 FF 01 0 F 00 01 01 00 00 00 00 00 01 00 00 00 00 02 00 60 00 04 00
00 00 00 00 14 00 FD 01 02 00 01 01 00 00 00 00 00 05 12 00 00 00 00 00 18 00 FF
01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02 00
01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00 00
05 20 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00
00 05 12 00 00 00
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\Type: 0x00000120
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\Start: 0x00000002
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\ErrorControl:
0x00000002
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\ImagePath:
"C:\WINDOWS \system32 \mfm \msrll.exe"
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\DisplayName: "Rll
enhanced drive"
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001 \Services \mfm\ObjectName:
"LocalSystem"
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet \Services \SharedAccess \Para
meters \FirewallPolicy \StandardProfile \AuthorizedApplications \List\C:\WINDOWS \syst
em32\mfm\msrll.exe: "C: \WINDOWS \system32 \mfm \msrll.exe:*:Enabled:msrll"
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet \Services \mfm\Security \Securi
ty: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00
00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 01 00 00 00 00 02 00 60 00 04
00 00 00 00 00 14 00 FD 01 02 00 01 01 00 00 00 00 00 05 12 00 00 00 00 00 18 00
FF 01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02
00 01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

00 05 20 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00 00 05 12 00 00 00
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm\Type: 0x00000120
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm\Start: 0x00000002
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm\ErrorControl: 0x00000002
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm\ImagePath: "C:\WINDOWS\system32\mfm\msrll.exe"
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\mfm\ObjectName: "LocalSystem"
HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\WINDOWS\system32\mfm\msrll.exe: "msrll"
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P:\Qbphzragf naq Frggvatf\whyvn\Qrfxgbc\zfeyy.rkr: 09 00 00 00 06 00 00 00 00 49 0C D6 2E CB C4 01
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\Documents and Settings\julia\Desktop\msrll.exe: "msrll"
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\WINDOWS\system32\mfm\msrll.exe: "msrll"
HKEY_USERS\S-1-5-18\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\WINDOWS\system32\mfm\msrll.exe: "msrll"

Values modified:7

HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Cryptography\RNG\Seed: 65 A8 FB 12 41 5C 75 3E A5 68 43 A3 5C E8 F8 BE DD 82 69 43 7D 90 C5 B8 58 EB EE D5 5C DA B3 B7 A6 34 1C F0 BF AD 5C 26 91 0A 6A 31 85 C2 71 E6 DF 53 3B 8B AE B6 C1 92 EE D0 0E 6C 95 C3 FB 1E 60 DC F5 F7 8D 74 DD 2E 55 8A 66 C2 D7 72 89 DD
HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Cryptography\RNG\Seed: FC 3C 45 E1 6A 76 3D 95 39 26 8A F0 B9 3A 9B B3 82 F9 A6 CB F7 C5 6C 2F 4D 92 F0 2D AC 02 E3 15 74 56 B3 95 B7 18 0C 63 2E 88 B5 93 40 83 C5 F9 3C AD 71 26 BF 1A 7C 41 72 11 44 78 AF 56 FC 9C FF AA 90 FB 33 C3 AF 1E 95 36 F6 25 87 E7 09 34
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001\Services\SharedAccess\Epoch\Epoch: 0x000000D8
HKEY_LOCAL_MACHINE \SYSTEM\ControlSet001\Services\SharedAccess\Epoch\Epoch: 0x000000DA
HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\SharedAccess\Epoch\Epoch: 0x000000D8

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\SharedAccess\Epoch\Epoch: 0x000000DA
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU: 09 00 00 00 DA 00 00 00 D0 4E 7D 62 2E CB C4 01
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU: 09 00 00 00 DB 00 00 00 00 49 0C D6 2E CB C4 01
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_HVFPHG: 09 00 00 00 52 00 00 00 D0 1F FE F7 2D CB C4 01
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_HVFPHG: 09 00 00 00 53 00 00 00 80 5A D5 D5 2E CB C4 01
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections\SavedLegacySettings: 3C 00 00 00 2D 00 00 00 0 1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 EF 51 F3 99 BB C4 01 01 00 00 00 C0 A8 2E 80 00 00 00 00 00 00 00 00
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections\SavedLegacySettings: 3C 00 00 00 2E 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 EF 51 F3 99 BB C4 01 01 00 00 00 C0 A8 2E 80 00 00 00 00 00 00 00 00
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\SessionInformation \ProgramCount: 0x00000004
HKEY_USERS\S-1-5-21-823518204-630328440-1417001333-1003\SessionInformation \ProgramCount: 0x00000005

Files added:5

C:\WINDOWS\Prefetch\MSRLL.EXE-03966588.pf
C:\WINDOWS\Prefetch\MSRLL.EXE-326D2A8E.pf
C:\WINDOWS\Prefetch\RUNDLL32.EXE-46508B14.pf
C:\WINDOWS\system32\mfm\jtram.conf
C:\WINDOWS\system32\mfm\msrll.exe

Files deleted:1

C:\Documents and Settings \julia\Desktop\msrll.exe

Files [attributes?] modified:8

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

C:\Documents and Settings \julia\Cookies\index.dat
C:\Documents and Settings \julia\Local Settings \History\History.IE5\index.dat
C:\Documents and Settings \julia\Local Settings \Temporary Internet Files\Content.IE5\index.dat
C:\Documents and Settings \julia\NTUSER.DAT.LOG
C:\WINDOWS\Prefetch\TASKMGR.EXE-20256C55.pf
C:\WINDOWS\system32\config\default.LOG
C:\WINDOWS\system32\config\software.LOG
C:\WINDOWS\system32\config\system.LOG

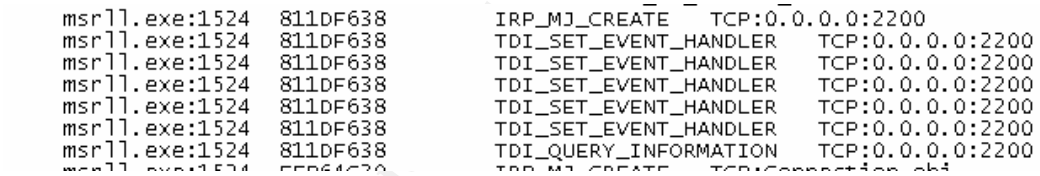
Folders added:3

C:\WINDOWS\system32\mfm
C:\WINDOWS\system32\mfm\.
C:\WINDOWS\system32\mfm\..

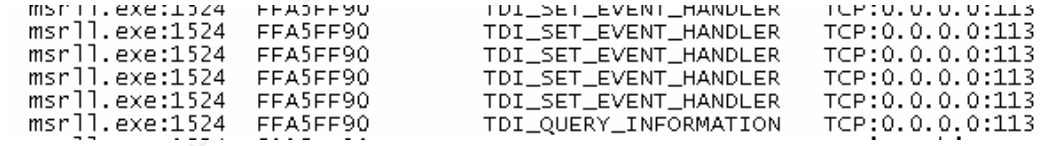
Total changes:49

TDIMon logs

Msrll.exe setting a listener on port 2200



Msrll.exe setting a listener on port 113



Msrll.exe sending a udp packet to 192.168.46.1:53 (via svchost.exe)

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Appendix B

Complete Bintext Output

```
File pos   Mem pos      ID   Text
========   =======     ==   ====

0000004D   0040004D     0   !This program cannot be run in DOS mode.
00000088   00400088     0   [AspackDie!]
00000178   00400178     0   .text
000001A0   004001A0     0   .data
000001F0   004001F0     0   .idata
00000218   00400218     0   .aspack
00000240   00400240     0   .adata
00001326   00401326     0   ?insmod
0000132E   0040132E     0   ?rmmod
00001335   00401335     0   ?lsmod
00001399   00401399     0   %s: <mod name>
000013A8   004013A8     0   %s: mod list full
000013BA   004013BA     0   %s: err: %u
000013C6   004013C6     0   mod_init
000013CF   004013CF     0   mod_free
000013D8   004013D8     0   %s: cannot init %s
000013EB   004013EB     0   %s: %s loaded (%u)
000013FE   004013FE     0   %s: mod allready loaded
00001416   00401416     0   %s:%s err %u
000015B5   004015B5     0   %s:%s not found
000015C5   004015C5     0   %s: unloading %s
000016AE   004016AE     0   [%u] : %s hinst:%x
00001712   00401712     0   unloading %s
000017A0   004017A0     0   %s: invalid_addr: %s
000017B5   004017B5     0   %s%s [port]
000018E8   004018E8     0   finished %s
00001A40   00401A40     0   %s <ip> <port> <t_time> <delay>
00001B32   00401B32     0   sockopt: %u
00001B3E   00401B3E     0   sendto err: %u
00001B4D   00401B4D     0   sockraw: %u
00001B59   00401B59     0   syn: done
00001FBC   00401FBC     0   %s <ip> <duration> <delay>
00002096   00402096     0   sendto: %u
000020A2   004020A2     0   jolt2: done
00002260   00402260     0   %s <ip> <p size> <duration> <delay>
00002356   00402356     0   Err: %u
0000235E   0040235E     0   smurf done
00002567   00402567     0   PhV#@
000025DE   004025DE     0   &err: %u
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
00002753   00402753     0   ?ping
00002763   00402763     0   ?smurf
0000276A   0040276A     0   ?jolt
00002820   00402820     0   PONG :%s
0000283A   0040283A     0   0h (@
0000299D   0040299D     0   %s!%s@%s
00002B3D   00402B3D     0   %s!%s
00002BB6   00402BB6     0   SVh=+@
00002BD7   00402BD7     0   irc.nick
00002BE0   00402BE0     0   NICK %s
00002EEA   00402EEA     0   NETWORK=
00002FF8   00402FF8     0   irc.pre
000032CC   004032CC     0   _%s__
000032D2   004032D2     0   __%s__
000032D9   004032D9     0   __%s___
000032E1   004032E1     0   NICK %s
000032F0   004032F0     0   %s %s
000036B0   004036B0     0   irc.chan
00003775   00403775     0   %s %s
0000377B   0040377B     0   WHO %s
000037C8   004037C8     0   PPhV,@

File pos   Mem pos      ID   Text
========   =======     ==   ====

00003A45   00403A45     0   USERHOST %s
00003A52   00403A52     0   logged into %s(%s) as %s
00003A97   00403A97     0   <$hE:@
00003ABB   00403ABB     0   PhR:@
00003B99   00403B99     0   nick.pre
00003BA2   00403BA2     0   %s -%04u
00003BAA   00403BAA     0   irc.user
00003BB3   00403BB3     0   irc.usereal
00003BBF   00403BBF     0   irc.real
00003BC8   00403BC8     0   irc.pass
00003BE0   00403BE0     0   tsend(): connection to %  s:%u failed
00003C20   00403C20     0   USER %s localhost 0 :%s
00003C38   00403C38     0   NICK %s
00003DF5   00403DF5     0   Ph <@
000040BF   004040BF     0   PRIVMSG
00004100   00404100     0   trecv(): Disconnected from %s err:%u
0000446B   00404 46B     0   NOTICE
00004472   00404472     0   %s %s :%s
00004615   00404615     0   Ph}D@
00004711   00404711     0   MODE %s  -o+b %s *@%s
00004798   00404798     0   C'PSWh
000047B4   004047B4     0   Sh'G@
000047E7   004047E7     0   MODE %s  -bo %s %s
0000487B   0040487B     0   Sh'G@
00004924   00404924     0   %s.key
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
00004A63   00404A63     0   Ph'G@
00004AA8   00404AA8     0   sk#%u %s is dead!
00004ABA   00404ABA     0   s_check: %s dead? pinging...
00004AD7   00404AD7     0   PING :ok
00004B00   00404B00     0   s_check: send error to %s disconnecting
00004B28   00404B28     0   expect the worst
00004B39   00404B39     0   s_check: killing socket %s
00004B54   00404B54     0   irc.knick
00004B5E   00404B5E     0   jtr.%u%s.iso
00004B6B   00404B6B     0   ison %s
00004B74   00404B74     0   servers
00004B7C   00404B7C     0   s_check: trying %s
00004DAA   00404DAA     0   Ph9K@
00004ED5   00404ED5     0   PhkK@
00004F41   00404F41     0   ShtK@
00004FD8   00404FD8     0   uYVh| K@
00005052   00405052     0   %s.mode
0000505A   0040505A     0   MODE %s %s
00005078   00405078     0   ShRP@
000050DA   004050DA     0   Sh$I@
000051A8   004051A8     0   PShZP@
000055A3   004055A3     0   mode %s +o %s
000055B2   004055B2     0   akick
000055B8   004055B8     0   mode %s +b %s %s
000055CA   004055CA     0   KICK %s %s
00005760   00405760     0   irc.pre
00005781   00405781     0   Set an irc sock to preform %s command on
000057AB   004057AB     0     Type
000057B3   004057B 3     0   %csklist
000057BC   004057BC     0    to view current sockets, then
000057DC   004057DC     0   %cdccsk
000057E4   004057E4     0   <#>
000058B4   004058B4     0   %s: dll loaded
000058C3   004058C3     0   %s: %d
0000597B   0040597B     0   RhHY@

File pos   Mem pos     ID   Text
========   =======     ==   ====

000059C6   004059C6     0   RhHY@
000059E1   004059E1     0   said %s to %s
000059EF   004059EF     0   usage: %s <target> "text"
00005A74   00405A74     0   %s not on %s
00005A81   00405A81     0   usage: %s <nick> <chan>
00005B20   00405B20     0   %s logged in
00005B87   00405B87     0   Sh [@
00005BA2   00405BA2     0   sys: %s bot: %s
00005BB2   00405BB2     0   preformance counter not avail
00005C2B   00405C2B     0   usage: %s <cmd>
00005C3B   00405C3B     0   %s free'd
```

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
00005C45   00405C45     0  unable to free %s
00005C6F   00405C6F     0  0h+ \@
00005CAD   00405CAD     0  later!
00005CB4   00405CB4     0  unable to %s errno:%u
00005D40   00405D40     0  service:%c user:%s inet connection:%c contype:%s
reboot privs:%c
00005E09   00405E09     0  Ph@]@
00005E23   00405E23     0  % -5u %s
00005F8F   00405F8F     0  %s: %s
00005F96   00405F96     0  %s: somefile
0000603F   0040603F     0  PhHY@
000060D4   004060D4     0  host: %s ip: %s
00006269   00406269     0  capGetDriverDescriptionA
00006292   00406292     0  cpus:%u
000062A0   004062A0     0  WIN%s (u:%s)%s%s mem:(%u/%u) %u%% %s %s
000065CB   004065CB     0  %s: %s (%u)
00006708   00406708     0  %s %s
00006754   00406754     0  %s bad args
000067BC   004067BC     0  3hTg@
000067DA   004067DA     0  akick
000067E8   004067E8     0  %s[%u] %s
000067F2   004067F2     0  %s removed
000067FD   004067FD     0  couldnt find  %s
0000680D   0040680D     0  %s added
00006816   00406816     0  %s allready in list
0000682A   0040682A     0  usage: %s +/ - <host>
0000696F   0040696F     0  7h*h@
000069EB   004069EB     0  jtram.conf
000069F6   004069F6     0  %s /t %s
000069FF   004069FF     0  jtr.home
00006A08   00406A08     0  %s \%s
00006A0E   00406A0E     0  %s: possibly failed: code %u
00006A2B   00406A2B     0  %s: possibly failed
00006A3F   00406A3F     0  %s: exec of %s failed err: %u
00006A90   00406A90     0  u.exf
00006C2D   00406C2D     0  Ph+j@
00006C82   00406C82     0  Ph?j@
00006CBC   00406CBC     0  jtr.id
00006CC3   00406CC3     0  %s: <url> <id>
00006ED7   00406ED7     0  IREG
00006EDD   00406EDD     0  CLON
00006EE3   00406EE3     0  ICON
00006EF8   00406EF8     0  WCON
00006F40   00406F40     0    #%u [fd:%u] %s:%u [%s%s] last:%u
00006F63   00406F63     0     | \=> [n:%s fh:%s] (%s)
00006F82   00406F82     0     | ---[%s] (%u) %s
00006F96   00406F96     0     |      |-[%s%s] [%s]
00006FAD   00406FAD     0      |=> (%s) (%.8x)
0000716E   0040716E     0  B$PRhco@
00007360   00407360     0  %s <pass> <salt>
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
File pos   Mem pos      ID   Text
========   =======     ==   ====

000073C8   004073C8     0   %s <nick> < chan>
0000748B   0040748B     0   PING %s
000074C9   004074C9     0   mIRC v6.12 Khaled Mardam -Bey
000074E7   004074E7     0   VERSION %s
0000751C   0040751C     0   dcc.pass
00007525   00407525     0   temp add %s
000075BD   004075BD     0   $h%u@
0000766A   0040766A     0   %s%u -%s
00007675   00407675     0   %s opened (%u)
000076A0   004076A0     0   %u bytes from %s in %u seconds saved to %s
000076CB   004076CB     0   (%s %s): incomplete! %u bytes
000076E9   004076E9     0   couldnt open %s  err:%u
00007700   00407700     0   (%s) %s: %s
0000770C   0040770C     0   (%s) urlopen failed
00007720   00407720     0   (%s): inetopen failed
00007798   00407798     0   Whjv@
00007B9D   00407B9D     0   Ph w@
00007BE4   00407BE4     0   no file n ame in %s
00007DDB   00407DDB     0   %s created
00007E49   00407E49     0   %s %s to %s Ok
00007E8F   00407E8F     0   3hl~@
00007EE0   00407EE0     0   %0.2u/%0.2u/%0.2u  %0.2u:%0.2u  %15s  %s
00007F09   00407F09     0   %s (err: %u)
0000806B   0040 806B    0   ShHY@
00008085   00408085     0   err: %u
000080F8   004080F8     0   %s %s :ok
00008165   00408165     0   unable to %s %s (err: %u)
000081C3   004081C3     0   ShHY@
000081F5   004081F5     0   % -16s %s
00008200   00408200     0   % -16s (%u.%u.%u.%u)
00008489   00408489     0   [%s][%s] %s
00008595   00408595     0   closing %u [%s:%u]
000085A8   004085A8     0   unable to close socket %u
000087E2   004087E2     0   using sock #%u %s:%u (%s)
000087FD   004087FD     0   Invalid soc k
0000880B   0040880B     0   usage %s <socks #>
000088D7   004088D7     0   leaves %s
000088E1   004088E1     0   :0 * * :%s
00008A96   00408A96     0   joins: %s
00008B82   00408B82     0   ACCEPT
00008B89   00408B89     0   resume
00008B90   00408 B90    0   err: %u
00008B99   00408B99     0   DCC ACCEPT %s %s %s
00008BAE   00408BAE     0   dcc_resume: cant find port %s
00008BD1   00408BD1     0   dcc.dir
00008BD9   00408BD9     0   %s \%s\%s\%s
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
00008BE5   00408BE5     0   unable to open (%s):   %u
00008BFD   00408BFD     0   resuming dcc from %s to %s
00008C19   00408C19     0   DCC RESUME %s %s %u
0000934E   0040934E     0   ?clone
00009355   00409355     0   ?clones
0000935D   0040935D     0   ?login
00009364   00409364     0   ?uptime
0000936C   0040936C     0   ?reboot
00009374   00409374     0   ?status
0000937C   0040937C     0   ?jump
00009382   00409382     0   ?nick
00009388   00409388     0   ?echo
0000938E   0040938E     0   ?hush
00009394   00409394     0   ?wget

File pos   Mem pos      ID   Text
========   =======      ==   ====

0000939A   0040939A     0   ?join
000093A9   004093A9     0   ?akick
000093B0   004093B0     0   ?part
000093B6   004093B6     0   ?dump
000093C6   004093C6     0   ?md5p
000093CC   004093C C      0   ?free
000093D7   004093D7     0   ?update
000093DF   004093DF     0   ?hostname
000093EE   004093EE     0   ?!fif
000093FE   004093FE     0   ?play
00009404   00409404     0   ?copy
0000940A   0040940A     0   ?move
00009415   00409415     0   ?sums
00009423   00409423     0   ?rmdir
0000942A   0040942A     0   ?mkdir
00009436   00409436     0   ?exec
00009440   00409440     0   ?kill
00009446   00409446     0   ?killall
0000944F   0040944F     0   ?crash
0000946E   0040946E     0   ?sklist
00009476   00409476     0   ?unset
0000947D   0040947D     0   ?uattr
00009484   00409484     0   ?dccsk
00009490   00409490     0   ?killsk
00009499   00409499     0   VERSION*
000094AE   004094AE     0   IDENT
000096BE   004096BE     0    %ud %02uh %02um %02us
000096D4   004096D4     0   %02uh %02um %02us
000096E6   004096E6     0   %um %02us
000099E0   004099E0     0   jtram.conf
000099EB   004099EB     0   jtr.*
000099F5   004099F5     0   DiCHFc2ioiVmb3cb4zZ7zWZH1oM=
```

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
00009A16   0040 9A16      0   conf_dump: wrote %u lines
0000A270   0040A270      0   get of %s incomplete at %u bytes
0000A2B0   0040A2B0      0   get of %s completed (%u bytes), %u seconds %u cps
0000A2F0   0040A2F0      0   error while writing to %s (%u)
0000A65C   0040 A65C      0   chdir: %s  -> %s (%u)
0000A750   0040A750      0   dcc_wait: get of %s from %s timed out
0000A790   0040A790      0   dcc_wait: closing [#%u] %s:%u (%s)
0000A9F0   0040A9F0      0   %4s #%.2u %s %ucps %u%% [sk#%u] %s
0000AA30   0040AA30      0    %u Send(s) %u Get(s) (%u transfer(s) total) UP:%ucps
DOWN:%ucps Total:%ucps
0000AC95   0040AC95      0   PRQh0
0000ACD0   0040ACD0      0    send of %s incomplete at %u bytes
0000AD10   0040AD10      0   send of %s completed (%u bytes), %u seconds %u
cps
0000AF50   0040AF50      0   cant open %s (err:%u) pwd:{%s}
0000AF70   0040AF70      0   DCC SEND %s %u %u %u
0000B751   0040B751      0   %s %s
0000B757   0040B757      0   %s exited with code %u
0000B76E   0040B76E      0   %s \%s
0000B774   0040B774      0   %s: %s
0000B77B   0040B77B      0   exec: Error:%u pwd:%s cmd:%s
0000BB40   0040BB40      0   dcc.pass
0000BB49   0040BB49      0   bot.port
0000BB52   0040BB52      0   %s bad pass from "%s"@%s
0000BCC9   0040BCC9      0    %s: connect from %s
0000BD33   0040BD33      0   jtr.bin
0000BD3B   0040BD3B      0   msrll.exe
0000BD45   0040BD45      0   jtr.home
0000BD57   0040BD57      0   jtr.id
0000BD63   0040BD63      0   irc.quit

File pos   Mem pos      ID   Text
========   =======     ==   ====

0000BD6E   0040BD6E      0   servers
0000BD80   0040BD80      0
collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:8080
0000BDCA   0040BDCA      0   irc.chan
0000BDD3   0040BDD3      0   #mils
0000BDE0   0040BDE0      0   $1$KZLPLKDf$W8kl8J  r1X8DOHZsmlp9qq0
0000BE20   0040BE20      0   $1$KZLPLKDf$55isA1ITvamR7bjAdBziX.
0000C02F   0040C02F      0   SSL_get_error
0000C03D   0040C03D      0   SSL_load_error_strings
0000C054   0040C054      0   SSL_library_init
0000C065   0040C065      0   SSLv3 _client_method
0000C079   0040C079      0   SSL_set_connect_state
0000C08F   0040C08F      0   SSL_CTX_new
0000C09B   0040C09B      0   SSL_new
0000C0A3   0040C0A3      0   SSL_set_fd
0000C0AE   0040C0AE      0   SSL_connect
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
0000C0BA   0040C0BA     0   SS L_write
0000C0C4   0040C0C4     0   SSL_read
0000C0CD   0040C0CD     0   SSL_shutdown
0000C0DA   0040C0DA     0   SSL_free
0000C0E3   0040C0E3     0   SSL_CTX_free
0000C263   0040C263     0   kernel32.dll
0000C270   0040C270     0   QueryPerformanceC  ounter
0000C288   0040C288     0   QueryPerformanceFrequency
0000C2A2   0040C2A2     0   RegisterServiceProcess
0000C2B9   0040C2B9     0   jtram.conf
0000C5B1   0040C5B1     0   irc.user
0000C5BA   0040C5BA     0   %s : USERID : UNIX : %s
0000C6A4   0040C6A4     0   QUIT :FUCK %u
0000C742   0040C742     0   Killed!? Arrg! [%u]
0000C756   0040C756     0   QUIT :%s
0000C7E8   0040C7E8     0   SeShutdownPrivilege
0000C888   0040C888     0   %s \%s
0000C88E   0040C88E     0   %s \%s\%s
0000C897   0040 C897    0   Rll enhanced drive
0000C8C0   0040C8C0     0   software \microsoft\windows\currentversion\run
0000C8EE   0040C8EE     0   /d "%s"
0000CE3D   0040CE3D     0   < u&
0000D010   0040D010     0
./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghi  jklmnopqrstuvwxyz
0000EA60   0040EA60     0   usage %s: server[:port] amount
0000EB33   0040EB33     0   %s: %s
0000EB3E   0040EB3E     0   %s %s %s <PARAM>
0000EB80   0040EB80     0   %s: [NETWORK|all] %s <"parm"> ...
0000EE20   0040EE20     0   USER  %s localhost 0 :%s
0000EE38   0040EE38     0   NICK %s
0000EEE4   0040EEE4     0   PSVh
0000F140   0040F140     0   md5.c
0000F146   0040F146     0   md != NULL
0000F8F1   0040F8F1     0   buf != NULL
0000F99F   0040F99F     0   hash != NULL
0000FAC5   0040FAC5     0   message digest
0000FAD4   0040FAD4     0   abcdefghijklmnopqrstuvwxyz
0000FB00   0040FB00     0
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
0000FB40   0040FB40     0
12345678901234567890123456789012345678901  23456789012345678901234567
8901234567890
0000FCE0   0040FCE0     0   sprng
0000FD11   0040FD11     0   sprng.c
0000FD19   0040FD19     0   buf != NULL
0000FDBC   0040FDBC     0   rc6.c
0000FDC2   0040FDC2     0   skey != NULL
0000FDCF   0040FDCF     0   key != NULL
0000FFD1   0040FFD1     0   ct != NULL
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
File pos   Mem pos      ID   Text
========   =======      ==   ====

0000FFDC   0040FFDC      0   pt != NULL
0001023E   0041023E      0   #4EVgx
00010256   00410256      0   $5FWhy
00010282   00410282      0   #4EVgx
0001029A   0041029A      0   $5FWhy
000102C6   004102C6      0   #4EVgx
000102DE   004102DE      0   $5FWhy
000102F8   004102F8      0   gN]HU
000103C3   004103C3      0   desired_keysize != NULL
00010430   00410430      0   ctr.c
00010436   00410436      0   ctr != NULL
00010442   00410442      0   key != NULL
0001044E   0041044E      0   count != NULL
00010546   00410546      0   ct != NULL
00010551   00410551      0   pt != NULL
000106F0   004106F0      0
ABCDEFGHIJKLMNOPQRSTUVWXYZabcd efghijklmnopqrstuvwxyz0123456789+/
0001077F   0041077F      0   ?456789:;<=
000107B7   004107B7      0    !"#$%&'()*+, -./0123
00010850   00410850      0   base64.c
00010859   00410859      0   outlen != NULL
00010868   00410868      0   out != NULL
00010874   00410874      0   in != NULL
00010B30   00410B30      0   _ARGCHK '%s' failure on line %d of file %s
00010B8B   00410B8B      0   crypt.c
00010B93   00410B93      0   name != NULL
00010D79   00410D79      0   cipher != NULL
00010E70   00410E70      0   hash != NULL
00010F7A   00410F7A      0   prng != NULL
000110F0   004110F0      0   LibTomCrypt 0.83
00011102   00411102      0   Endianess: little (32 -bit words)
00011123   00411123      0   Clean stack: disabled
00011139   00411139      0   Ciphers buil t-in:
0001114B   0041114B      0     Blowfish
00011157   00411157      0     RC2
0001115E   0041115E      0     RC5
00011165   00411165      0     RC6
0001116C   0041116C      0     Serpent
00011177   00411177      0     Safer+
00011181   00411181      0     Safer
0001118A   0041118A      0     Rijndael
00011196   00411196      0     XTEA
0001119E   0041119E      0     Twofish
000111AA   004111AA      0     CAST5
000111B3   004111B3      0     Noekeon
000111BF   004111BF      0   Hashes built -in:
000111D0   004111D0      0     SHA -512
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
000111DB   004111DB     0     SHA -384
000111E6   004111E6     0     SHA -256
000111F1   004111F1     0     TIGER
000111FA   004111FA     0     SHA1
00011202   00411202     0     MD5
00011209   00411209     0     MD4
00011210   00411210     0     MD2
00011218   00411218     0   Block Chaining Modes:
0001122E   0041122E     0     CFB
00011235   00411235     0     OFB
0001123C   0041123C     0     CTR
00011244   00411244     0   PRNG:
0001124A   0041124A     0     Yarrow
00011254   00411254     0     SPRNG

File pos   Mem pos      ID   Text
========   =======      ==   ====

0001125D   0041125D     0     RC4
00011265   00411265     0   PK Algs:
0001126E   0041126E     0     RSA
00011275   00411275     0     DH
0001127B   0041127B     0     ECC
00011282   00411282     0     KR
00011289   00411289     0   Compiler:
00011293   00411293     0     WIN32 platform detected.
000112AF   004112AF     0     GCC compiler detected.
000112CA   004112CA     0   Various others:  BASE64  MPI  HMAC
00011313   00411313     0   /dev/random
00011430   00411430     0   Microsoft Base Cryptographic Provider v1.0
000114D2   004114D2     0   bits.c
000114D9   004114D9     0   buf != NULL
000114F6   004114F6     0   t9VWS
0001154A   0041154A     0   prng != NULL
00011832   00411832     0   <"tx< tf<       t
00011846   00411846     0   < tV<           t
00011852   00411852     0   < tJ<           tF
00011A10   00411A10     0    -LIBGCCW32-EH-SJLJ-GTHR-MINGW32
000130B0   004130B0     0    <ip> <total secs> <p size> <delay>
00013350   00413350     0   modem
00013358   00413358     0   Lan
0001335E   0041335E     0   Proxy
0001336B   0041336B     0   none
00013390   00413390     0   m220 1.0 #2730 Mar 16 11:47:38   2004
000133D4   004133D4     0   unable to %s %s (err: %u)
00013420   00413420     0   unable to kill %s (%u)
00013437   00413437     0   %s killed (pid:%u)
00013470   00413470     0   AVICAP32.dll
0001347D   0041347D     0   unable to kill %u (%u)
00013494   00413494     0   pid %u killed
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
000134A2   004134A2     0   error!
000134A9   004134A9     0   ran ok
000134B0   004134B0     0   MODE %s +o %s
000134BF   004134BF     0   set %s %s
00013600   00413600     0   Mozilla/4.0
0001360C   0041360C     0   Accept: */*
0001361C   0041361C     0   <DIR>
0001362B   0041362B     0   Could not copy %s to %s
00013643   00413643     0   %s copied to %s
00013653   00413653     0   0123456789abcdef
00013664   00413664     0   %s unset
0001366D   0041366D     0   unable to unset %s
00013AD4   00413AD4     0   (%s) %s
00013ADD   00413ADD     0   %s %s
00013BA0   00413BA0     0   libssl32.dll
00013BAD   00413BAD     0   libeay32.dll
00013BE0   00413BE0     0   <die|join|part|raw|msg>
0011B67A   0051B67A     0   AdjustTokenPrivileges
0011B692   0051B692     0   CloseServiceHandle
0011B6AA   0051B6AA     0   CreateServiceA
0011B6BE   0051B6BE     0   CryptAcquireContextA
0011B6D6   0051B6D6     0   CryptGenRandom
0011B6EA   0051B6EA     0   CryptReleaseContext
0011B702   0051B702     0   GetUserNameA
0011B712   0051B712     0   LookupPrivilegeValueA
0011B72A   0051B72A     0   OpenProcessToken
0011B73E   0051B73E     0   OpenSCManagerA
0011B752   0051B752     0   RegCloseKey

File pos   Mem pos      ID   Text
========   =======     ==   ====

0011B762   0051B762     0   RegCreateKeyExA
0011B776   0051B776     0   RegSetValueExA
0011B78A   0051B78A     0   RegisterServiceCtrlHandlerA
0011B7AA   0051B7AA     0   SetServiceStatus
0011B7BE   0051B7BE     0   StartServiceCtrlDispatcherA
0011B7DE   0051B7DE     0   AddAtomA
0011B7EA   0051B7EA     0   CloseHandle
0011B7FA   0051B7FA     0   CopyFileA
0011B806   0051B806     0   CreateDirectoryA
0011B81A   0051B81A     0   CreateFil eA
0011B82A   0051B82A     0   CreateMutexA
0011B83A   0051B83A     0   CreatePipe
0011B84A   0051B84A     0   CreateProcessA
0011B85E   0051B85E     0   CreateToolhelp32Snapshot
0011B87A   0051B87A     0   DeleteFileA
0011B88A   0051B88A     0   Dup licateHandle
0011B89E   0051B89E     0   EnterCriticalSection
0011B8B6   0051B8B6     0   ExitProcess
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
0011B8C6   0051B8C6      0   ExitThread
0011B8D6   0051B8D6      0   FileTimeToSystemTime
0011B8EE   0051B8EE      0   FindAtomA
0011B8FA   0051B8FA      0   FindClose
0011B906   0051B906      0   FindFirstFileA
0011B91A   0051B91A      0   FindNextFileA
0011B92A   0051B92A      0   FreeLibrary
0011B93A   0051B93A      0   GetAtomNameA
0011B94A   0051B94A      0   GetCommandLineA
0011B95E   0051B95E      0   GetCurrentDirectoryA
0011B976   0051B976      0   GetCurrentProcess
0011B98A   0051B98A      0   GetCurrentThreadId
0011B9A2   0051B9A2      0   GetExitCodeProcess
0011B9BA   0051B9BA      0   GetFileSize
0011B9CA   0051B9CA      0   GetFullPathNameA
0011B9DE   0051B9DE      0   GetLastError
0011B9EE   0051B9EE      0   GetModuleFileNameA
0011BA06   0051BA06      0   GetModuleHandleA
0011BA1A   0051BA1A      0   GetProcAddress
0011BA2E   0051BA2E      0   GetStartupInfoA
0011BA42   0051BA42      0   Get SystemDirectoryA
0011BA5A   0051BA5A      0   GetSystemInfo
0011BA6A   0051BA6A      0   GetTempPathA
0011BA7A   0051BA7A      0   GetTickCount
0011BA8A   0051BA8A      0   GetVersionExA
0011BA9A   0051BA9A      0   GlobalMemoryStatus
0011BAB2   0051BAB2      0   InitializeCriticalSection
0011BACE   0051BACE      0   IsBadReadPtr
0011BADE   0051BADE      0   LeaveCriticalSection
0011BAF6   0051BAF6      0   LoadLibraryA
0011BB06   0051BB06      0   MoveFileA
0011BB12   0051BB12      0   OpenProcess
0011BB22   0051BB22      0   PeekNamedPipe
0011BB32   0051BB32      0   Process32First
0011BB46   0051BB46      0   Process32Next
0011BB56   0051BB56      0   QueryPerformanceFrequency
0011BB72   0051BB72      0   ReadFile
0011BB7E   0051BB7E      0   ReleaseMutex
0011BB8E   0051BB8E      0   RemoveDirectoryA
0011BBA2   0051BBA2      0   SetConsoleCtrlHandler
0011BBBA   0051BBBA      0   SetCurrentDirectoryA
0011BBD2   0051BBD2      0   SetFilePointer

File pos   Mem pos      ID   Text
========   =======     ==   ====

0011BBE6   0051BBE6      0   SetUnhandledExceptionFilter
0011BC06   0051BC06      0   Sleep
0011BC0E   0051BC0E      0   TerminateProcess
0011BC22   0051BC22      0   WaitForSingleObject
```


© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
0011BC3A  0051BC3A   0  WriteFile
0011BC46  0051BC46   0  _itoa
0011BC4E  0051BC4E   0  _stat
0011BC56  0051BC56   0  _strdup
0011BC62  0051BC62   0  _stricmp
0011BC6E  0051BC6E   0  __getmainargs
0011BC7E  0051BC7E   0  __p__environ
0011BC8E  0051BC8E   0  __p__fmode
0011BC9E  0051BC9E   0  __set_app_type
0011BCB2  0051BCB2   0  _beginthread
0011BCC2  0051BCC2   0  _cexit
0011BCCE  0051BCCE   0  _errno
0011BCDA  0051BCDA   0  _fileno
0011BCEE  0051BCEE   0  _onexit
0011BCFA  0051BCFA   0  _setm ode
0011BD06  0051BD06   0  _vsnprintf
0011BD16  0051BD16   0  abort
0011BD1E  0051BD1E   0  atexit
0011BD32  0051BD32   0  clock
0011BD3A  0051BD3A   0  fclose
0011BD46  0051BD46   0  fflush
0011BD52  0051BD52   0  fgets
0011BD5A  0051BD5A   0  fopen
0011BD62  0051BD62   0  fprintf
0011BD6E  0051BD6E   0  fread
0011BD7E  0051BD7E   0  fwrite
0011BD8A  0051BD8A   0  malloc
0011BD96  0051BD96   0  memcpy
0011BDA2  0051BDA2   0  memset
0011BDAE  0051BDAE   0  printf
0011BDBA  0051BDBA   0  raise
0011BDCA  0051BDCA   0  realloc
0011BDD6  0051BDD6   0  setvbuf
0011BDE2  0051BDE2   0  signal
0011BDEE  0051BDEE   0  sprintf
0011BDFA  0051BDFA   0  srand
0011BE02  0051BE02   0  strcat
0011BE0E  0051BE0E   0  strchr
0011BE1A  0051BE1A   0  strcmp
0011BE26  0051BE26   0  strcpy
0011BE32  0051BE32   0  strerror
0011BE3E  0051BE3E   0  strncat
0011BE4A  0051BE4A   0  strncmp
0011BE56  0051BE56   0  strncpy
0011BE62  0051BE62   0  strstr
0011BE76  0051BE76   0  toupper
0011BE82  0051BE82   0  ShellExecuteA
0011BE92  0051BE92   0  DispatchMessageA
0011BEA6  0051BEA6   0  ExitWindowsEx
0011BEB6  0051BEB6   0  GetMessageA
```

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
0011BEC6   0051BEC6     0   PeekMessageA
0011BED6   0051BED6     0   GetFileVersionInfoA
0011BEEE   0051BEEE     0   VerQueryValueA
0011BF02   0051BF02     0   InternetCloseHandle
0011BF1A   0051BF1A     0   InternetGetConnectedSta  te
0011BF36   0051BF36     0   InternetOpenA

File pos   Mem pos      ID   Text
========   =======     ==   ====

0011BF46   0051BF46     0   InternetOpenUrlA
0011BF5A   0051BF5A     0   InternetReadFile
0011BF6E   0051BF6E     0   WSAGetLastError
0011BF82   0051BF82     0   WSASocketA
0011BF92   0051BF92     0   WSAStartup
0011BFA2   0051BFA2     0   __WSAFDIsSet
0011BFB2   0051BFB2     0   accept
0011BFC6   0051BFC6     0   closesocket
0011BFD6   0051BFD6     0   connect
0011BFE2   0051BFE2     0   gethostbyaddr
0011BFF2   0051BFF2     0   gethostbyname
0011C002   0051C002     0   gethostname
0011C012   0051C012     0   getsockname
0011C022   0051C022     0   htonl
0011C02A   0051C02A     0   htons
0011C032   0051C032     0   inet_addr
0011C03E   0051C03E     0   inet_ntoa
0011C04A   0051C04A     0   ioctlsocket
0011C05A   0051C05A     0   listen
0011C066   0051C066     0   ntohl
0011C076   0051C076     0   select
0011C08A   0051C08A     0   sendto
0011C096   0051C096     0   setsoc kopt
0011C0A6   0051C0A6     0   shutdown
0011C0B2   0051C0B2     0   socket
0011C0FC   0051C0FC     0   ADVAPI32.DLL
0011C1FC   0051C1FC     0   KERNEL32.dll
0011C21C   0051C21C     0   msvcrt.dll
0011C2E0   0051C2E0     0   msvcrt.dll
0011C2F0   0051C2F0     0   SHELL32.DLL
0011C30C   0051C30C     0   USER32.dll
0011C320   0051C320     0   VERSION.dll
0011C340   0051C340     0   WININET.DLL
0011C3B4   0051C3B4     0   WS2_32.DLL
0011D071   0051D071     0   VirtualAlloc
0011D07E   0051D07E     0   VirtualFree
0011D441   0051D441     0   kernel32.dll
0011D44E   0051D44E     0   ExitProcess
0011D45A   0051D45A     0   user32.dll
0011D465   0051D465     0   MessageBoxA
```

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

```
0011D471   0051D471      0   wsprintfA
0011D47B   0051D47B      0   LOADER   ERROR
0011D488   0051D488      0   The procedure entry point %s could not be located in
the dynamic link library %s
0011D4D9   0051D4D9      0   The ordinal %u could not be located in the dynamic
link library %s
0011D6E6   0051D6E6      0   (08@P
0011D874   0051D874      0   D4l|M
0011D9C0   0051D9C0      0   ;;F,s
0011D9CF   0051D9CF      0   ,;F0s
0011D9DB   0051D9DB      0   ;F4s
0011DCB5   0051DCB5      0   D$$W3
0011DF6C   0051DF6C      0   kernel32.dll
0011DF7B   0051DF7B      0   GetProcAddress
0011DF8C   0051DF8C      0   GetModuleHandleA
0011DF9F   0051DF9F      0   LoadLibraryA
0011E074   0051E074      0   advapi32.dll
0011E081   0051E081      0   msvcrt.dll
0011E08C   0051E08C      0   msvcrt.dll
0011E097   0051E097      0   shell32.dll
0011E0A3   0051E0A3      0   user32.dll
0011E0AE   0051E0AE      0   version.dll

File pos   Mem pos      ID   Text
========   =======     ==   ====

0011E0BA   0051E0BA      0   wininet.dll
0011E0C6   0051E0C6      0   ws2_32.dll
0011E113   0051E113      0   AdjustTokenPrivileges
0011E12B   0051E12B      0   _itoa
0011E133   0051E133      0   __getmainargs
0011E143   0051E143      0   ShellExecuteA
0011E153   0051E153      0   DispatchMessageA
0011E166   0051E166      0   GetFileVersionInfoA
0011E17C   0051E17C      0   InternetCloseHandle
0011E192   0051E192      0   WSAGetLastError
```

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Appendix C

Encryption/Randomisation -related strings

String	Description
%s <pass><salt>	a salt is usually a couple of random characters that are used by an encryption r outine to randomise the output.
/dev/random	"this is a little character device that gives you random numbers when you read it." Many encryption routines "use this device to seed a secure random number generator" [3]. However, it is not present on MS Operati ng systems.
DiCHFc2ioiVmb3cb4zZ7zWZH 1oM=	a hard-coded string that looks as if it is encrypted or maybe it is a key used by the encryption routine.
"1KZLPLKDf$W8kI8Jr1X8D OHZsmIp9qq0" "1KZLPLKDf$55isA1ITvam R7bjAdBziX."	More hard-coded strings that se em to be encrypted. Interestingly the first part of each string is the same - 1KZLPLKDf$ - also, the dollar signs seem to split the string up into sections.
./0123456789ABCDEFGHIJKL MNOPQRSTUVWXYZabcdefg hijklmnopqrstuvwxyz	This and similar strings coul d be used as part of the encryption routine.
sprng, sprng.c	A set of libraries for scalable and portable pseudorandom number generators.
rc6.c	An encryption algorithm
skey, key, desired_keysize	
crypt.c, cipher, hash	
LibTomCrypt 0.83, Endianess: little (32-bit words) Clean stack: disabled Ciphers built-in: Blowfish RC2 RC5 RC6 Serpent Safer+ Safer	A "cryptographic toolkit that provides developers with a vast array of well known published block ciphers, one-way hash functions, chaining modes, pseudo-random number generators, public -key cryptography and a plethora of other routines." [4] I found so many strings relating to this program that I believe that LibTomCrypt is packaged up with the malware.

[3] http://egd.sourceforge.net , EGD: The Entropy Gathering Daemon b y Brian Warner
[4] http://libtomcrypt.org/features.html

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Rijndael XTEA Twofish CAST5 Noekeon Hashes built-in: SHA-512 SHA-384 SHA-256 TIGER SHA1 MD5 MD4 MD2 Block Chaining Modes: CFB OFB CTR Yarrow SPRNG RC4 PK Algs: RSA ECC Compiler: WIN32 platform detected. GCC compiler detected. Various others: BASE64 MPI HMAC	
Microsoft Base Cryptographic Provider v1.0	A Cryptographic Service Provider is an independent software module that performs cryptography algorithms for authentication, encoding and encryption. This one comes with Internet Explorer 3.0 or later.
CryptAcquireContextA CryptGenRandom CryptReleaseContext	These three functions belong to ADVAPI32.dll. This dll name can also be found in the strings.

© SANS Institute 2005, Author retains full rights.

Attack-related strings

String	Description
jolt2: done,	Jolt is the name of a DoS attack which affects Windows 95 and NT machines. [5] "The attack sends very large, fragmented ICMP packets to a target

[5] http://www.physnet.uni -hamburg.de/physnet/security/vulnerability/jolt.html

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

	machine." The packets "are fragmented in such a way that the target machine is unable to reassemble them for use."
smurf done, ?smurf	A Smurf attack is a network level attack against hosts.[6] An attacker sends a large amount of ICMP traffic to a broadcast address, spoofing its source address to look like its coming from the victim. All the hosts on the network (that receive the ping) respond to the victim, causing a denial of service.
syn: done	A Syn attack is when an attacker sends lots of Synchronization packets to a victim, saturating the network and causing a denial of service.
?reboot, ?rmdir ?mkdir, ?crash ?sklist, ?dccsk ?killsk	A selection of what could be irc commands which look as if they could cause some trouble.

SSL-related strings

String	Description
SSL_get_error SSL_load_error_strings SSL_library_init SSLv3_client_method SSL_set_connect_state SSL_CTX_new SSL_new SSL_set_fd SSL_connect SSL_write SSL_read SSL_shutdown SSL_free SSL_CTX_free	SSL is Secure Sockets Layer. It is used for secure communication via the WWW. Maybe the malware is able to access a web server to receive instructions or maybe downlo ad files or updates.

DCC and IRC Socket-related strings

String	Description
"dcc_wait: get of %s from %s timed out" "dcc_wait: closing [#%u]	DCC is Direct-Client-to-Client. You can send an receive files over IRC with this function. You can also chat directly, privately and securely to

[6] http://www.pentics.net/denial-of-service/white-papers/smurf.cgi

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

%s:%u (%s)" "%u Send(s) %u Get(s) (%u transfer(s) total) UP:%ucps DOWN:%ucps Total:%ucps" "send of %s incomplet e at %u bytes" "send of %s completed (%u bytes), %u seconds %u cps" "cant open %s (err:%u) pwd:{%s}" "DCC SEND %s %u %u %u" "dcc.pass" ?dccsk DCC ACCEPT %s %s %s dcc_resume: cant find port %s dcc.dir %s\%s\%s\%s unable to open (%s): %u resuming dcc from % s to %s DCC RESUME %s %s %u	someone on IRC. It does not use chat channels to transmit information, rather it fo rms a direct link between two users. The strings found by Bintext indicate the Trojan has the capability to transferdata. There may also be a password involved ("dcc.pass"). Maybe the malware is using port 2200 to receive data from the attacker.
Set an irc sock to preform %s command on Type %csklist to view current sockets, then %cdccsk	

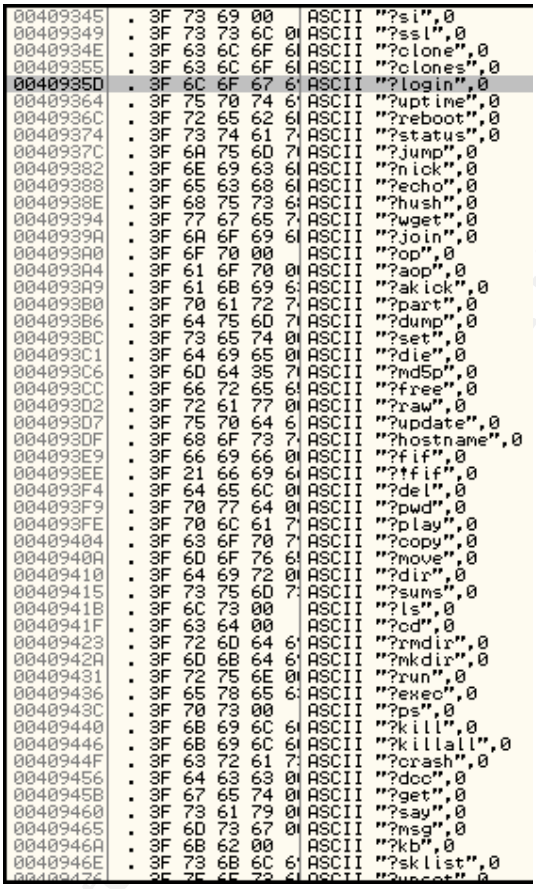
Strings related to attacker updating the malware specimen

String	Description
?insmod ?rmmod ?lsmod %s: <mod name> %s: mod list full mod_init mod_free %s: %s loaded (%u) %s: mod allready loaded %s: unloading %s %s:%s not found finished %s	The attacker may use these commands to check to see if the malware specimen already has a particular module. The attacker may be able to upload a module if necessary or remo ve a module.

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Appendix D

List of irc commands found in OllyDbg, including ?login



```
00409345  . 3F 73 69 00     ASCII "?si",0
00409349  . 3F 73 73 6C 0   ASCII "?ssl",0
0040934E  . 3F 63 6C 6F 6   ASCII "?clone",0
00409355  . 3F 63 6C 6F 6   ASCII "?clones",0
0040935D  . 3F 6C 6F 67 6   ASCII "?login",0
00409364  . 3F 75 70 74 6   ASCII "?uptime",0
0040936C  . 3F 72 65 62 6   ASCII "?reboot",0
00409374  . 3F 73 74 61 7   ASCII "?status",0
0040937C  . 3F 6A 75 6D 7   ASCII "?jump",0
00409382  . 3F 6E 69 63 6   ASCII "?nick",0
00409388  . 3F 65 63 68 6   ASCII "?echo",0
0040938E  . 3F 68 75 73 6   ASCII "?hush",0
00409394  . 3F 77 67 65 7   ASCII "?wget",0
0040939A  . 3F 6A 6F 69 6   ASCII "?join",0
004093A0  . 3F 6F 70 00     ASCII "?op",0
004093A4  . 3F 61 6F 70 0   ASCII "?aop",0
004093A9  . 3F 61 6B 69 6   ASCII "?akick",0
004093B0  . 3F 70 61 72 7   ASCII "?part",0
004093B6  . 3F 64 75 6D 7   ASCII "?dump",0
004093BC  . 3F 73 65 74 0   ASCII "?set",0
004093C1  . 3F 64 69 65 0   ASCII "?die",0
004093C6  . 3F 6D 64 35 7   ASCII "?md5p",0
004093CC  . 3F 66 72 65 6   ASCII "?free",0
004093D2  . 3F 72 61 77 0   ASCII "?raw",0
004093D7  . 3F 75 70 64 6   ASCII "?update",0
004093DF  . 3F 68 6F 73 7   ASCII "?hostname",0
004093E9  . 3F 66 69 66 0   ASCII "?fif",0
004093EE  . 3F 21 66 69 6   ASCII "?!fif",0
004093F4  . 3F 64 65 6C 0   ASCII "?del",0
004093F9  . 3F 70 77 64 0   ASCII "?pwd",0
004093FE  . 3F 70 6C 61 7   ASCII "?play",0
00409404  . 3F 63 6F 70 7   ASCII "?copy",0
0040940A  . 3F 6D 6F 76 6   ASCII "?move",0
00409410  . 3F 64 69 72 0   ASCII "?dir",0
00409415  . 3F 73 75 6D 7   ASCII "?sums",0
0040941B  . 3F 6C 73 00     ASCII "?ls",0
0040941F  . 3F 63 64 00     ASCII "?cd",0
00409423  . 3F 72 6D 64 6   ASCII "?rmdir",0
0040942A  . 3F 6D 6B 64 6   ASCII "?mkdir",0
00409431  . 3F 72 75 6E 0   ASCII "?run",0
00409436  . 3F 65 78 65 6   ASCII "?exec",0
0040943C  . 3F 70 73 00     ASCII "?ps",0
00409440  . 3F 6B 69 6C 6   ASCII "?kill",0
00409446  . 3F 6B 69 6C 6   ASCII "?killall",0
0040944F  . 3F 63 72 61 7   ASCII "?crash",0
00409456  . 3F 64 63 63 0   ASCII "?dcc",0
0040945B  . 3F 67 65 74 0   ASCII "?get",0
00409460  . 3F 73 61 79 0   ASCII "?say",0
00409465  . 3F 6D 73 67 0   ASCII "?msg",0
0040946A  . 3F 6B 62 00     ASCII "?kb",0
0040946E  . 3F 73 6B 6C 6   ASCII "?sklist",0
```

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Appendix E

ADIPro ScreenShots

Subroutine 404481

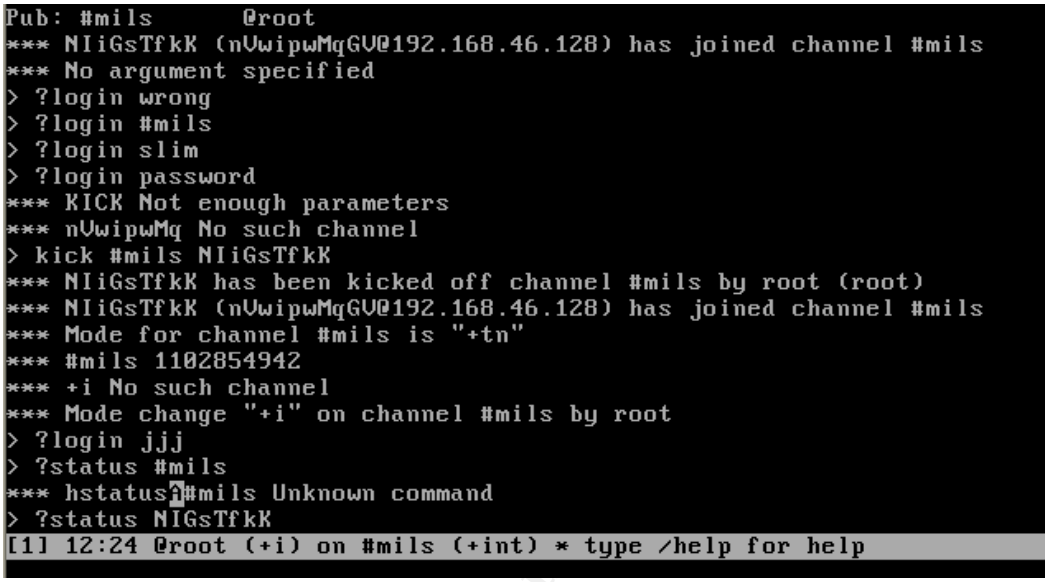
```
text:00404481                push    ebp
text:00404482                mov     ebp, esp
text:00404484                push    edi
text:00404485                push    esi
text:00404486                push    ebx
text:00404487                mov     eax, 101Ch
text:0040448C                call    sub_411D10
text:00404491                mov     esi, [ebp+arg_8]
text:00404494                mov     [ebp+var_101C], 0
text:0040449E                test    byte ptr [esi+2058h], 1
text:004044A5                jz      short loc_4044D2
text:004044A7                cmp     dword ptr [esi], 0FFFFFFFFh
text:004044AA                jz      short loc_4044D2
text:004044AC                sub     esp, 0Ch
text:004044AF                push    8
text:004044B1                call    malloc
text:004044B6                mov     [ebp+var_1020], eax
text:004044BC                add     esp, 10h
text:004044BF                test    eax, eax
text:004044C1                jnz     short loc_4044DC
text:004044C3                sub     esp, 0Ch
text:004044C6                lea     eax, [ebp+var_1018]
text:004044CC                push    eax
```

© SANS Institute 2005

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.

Appendix F

IRC screen shots



© SANS Institute 2005, Au

© SANS Institute 2005, As part of GIAC practical repository Author retains full rights.