



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 permitted without express written permission.

Interested in learning more?

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

Richard Wanner
GIAC Reverse Engineering Malware (GREM)
Version 1.0 (July 2004)

NS2004
Submitted: February 23, 2005

Reverse Engineering msrll.exe

© SANS Institute 2005, Author retains full rights.

Contents

<u>Abstract</u>	5
<u>Laboratory Setup</u>	5
<u>Static Analysis</u>	8
<u>Properties of the Malware specimen</u>	8
<u>Embedded strings</u>	9
<u>Behavioural Analysis</u>	11
<u>First Iteration</u>	13
<u>Second Iteration</u>	20
<u>Third Iteration</u>	21
<u>Code Analysis</u>	23
<u>Analyzing the Command Input via Port 2200</u>	26
<u>Analyzing the jtram.conf File</u>	32
<u>Analysis Wrap-Up</u>	34
<u>Appendix A: Tools Utilized</u>	36
<u>Windows Tools</u>	36
<u>Unix Tools</u>	36
<u>Appendix B – Active Ports Baseline</u>	37
<u>Appendix C Nmap Baselines</u>	38
<u>TCP Baseline</u>	38
<u>UDP Baseline</u>	38
<u>Appendix D Process Explorer Baseline</u>	39
<u>Appendix E Summary of Interesting Strings Output</u>	40
<u>Appendix F 6667/9999/8080 Connection Attempts</u>	41
<u>Appendix G: Malware IRC Connection</u>	46
<u>Appendix H BinText output for unpacked malware</u>	53
<u>Appendix I Commands Test</u>	68

Figures

<u>Figure 1: High Level Lab Architecture</u>	5
<u>Figure 2: Virtual Machine Virtual Network Setup</u>	7
<u>Figure 3: Malware Size</u>	9
<u>Figure 4: Malware md5sum</u>	9
<u>Figure 5: Regshot interface</u>	11
<u>Figure 6: Sysinternals tools interface</u>	12
<u>Figure 7: Regshot 2nd shot</u>	13
<u>Figure 8: msrll.exe - Process Explorer</u>	14
<u>Figure 9: Filemon Output – First Iteration</u>	14
<u>Figure 10: C:\Windows\system32\mfm creation and move of msrll.exe</u>	15
<u>Figure 11: C:\Windows\system32\mfm\msrll.exe size and md5sum</u>	15

<u>Figure 12: Filemon - Creation of jtram.conf</u>	16
<u>Figure 12: C:\Windows\system32\mfem\jtram.conf size and md5sum</u>	16
<u>Figure 13: jtram.conf file</u>	17
<u>Figure 14: netcat output</u>	19
<u>Figure 15: nmap output</u>	20
<u>Figure 16: jtram.conf size and md5sum</u>	20
<u>Figure 17: ping collective7.zxy0.com</u>	21
<u>Figure 18: Malware Connects to IRC Server</u>	21
<u>Figure 19: Malware joins IRC channel #mils</u>	22
<u>Figure 20: Malware in the IRC channel #mils</u>	22
<u>Figure 21: Attempting to communicate with the malware on port 2200</u>	22
<u>Figure 22: IDA Pro warning message</u>	23
<u>Figure 23: .aspack prefix on imported code</u>	24
<u>Figure 24: AspackDie success message</u>	24
<u>Figure 25: Unpacked malware size and md5sum</u>	25
<u>Figure 26: Xrefs to recv()</u>	26
<u>Figure 27: After recv() breakpoints</u>	26
<u>Figure 28: Calls to sub_40E66A</u>	27
<u>Figure 29: Input string pointer in EDX</u>	27
<u>Figure 30: Jump Equal in input routine</u>	28
<u>Figure 31: "notpasswdnotpasswd" input to netcat</u>	29
<u>Figure 32: Stacked parameters for msrll.00405872</u>	29
<u>Figure 33: Stacked parameters for msrll.0040D611</u>	29
<u>Figure 34: Stacked parameters for msrll.0040D07E</u>	29
<u>Figure 35: Hashes in registers</u>	30
<u>Figure 36: Test for valid password</u>	30
<u>Figure 37: JE replaced with NOPs</u>	30
<u>Figure 38: ?status - Response from the malware</u>	30
<u>Figure 39: ?status output string</u>	32
<u>Figure 40: xrefs to ?status output string</u>	32
<u>Figure 41: jtram.conf CreateFileA</u>	32
<u>Figure 42: Zero length jtram.conf file</u>	33
<u>Figure 43: jtram.conf WriteFile</u>	33
<u>Figure 44: msrll.0040B2B0 stack parameters</u>	33

Tables

<u>Table 1: Virtual Machine Configurations</u>	6
<u>Table 2: msrll.exe Referenced DLLs</u>	10
<u>Table 3: Functions in Strings Output</u>	10
<u>Table 4: Properties of the Malware Specimen</u>	11
<u>Table 6: msrll.exe Added as a Service</u>	17
<u>Table 7: msrll.exe Service Subkey Descriptions</u>	18
<u>Table 8: TDIMON - Port 113 Open</u>	19
<u>Table 9: TDIMON - Port 2200 Open</u>	19
<u>Table 10: Potential Malware Commands</u>	25
<u>Table 11: Possible Password Strings</u>	25
<u>Table 12: jtram.conf contents</u>	33

© SANS Institute 2005, Author retains full rights.

Abstract

This paper describes the reverse engineering of a piece of Windows malware called msrll.exe. The paper describes the laboratory setup used in the malware lab and the tools and methodology used in the malware analysis. A static analysis is performed on the malware to determine the characteristics of the malware. A behavioural analysis is also carried out to determine the behavioural characteristics of the malware and how it affects and interacts with the Windows host. A code analysis is carried out to determine the subtler aspects of the malware. Finally, the testing is summarized and suggestions for preventing the infection from this sort of malware, and how to mitigate the impact of the malware if you are infected.

Laboratory Setup

The diagram below shows a high level view of the laboratory setup utilized for this assignment.

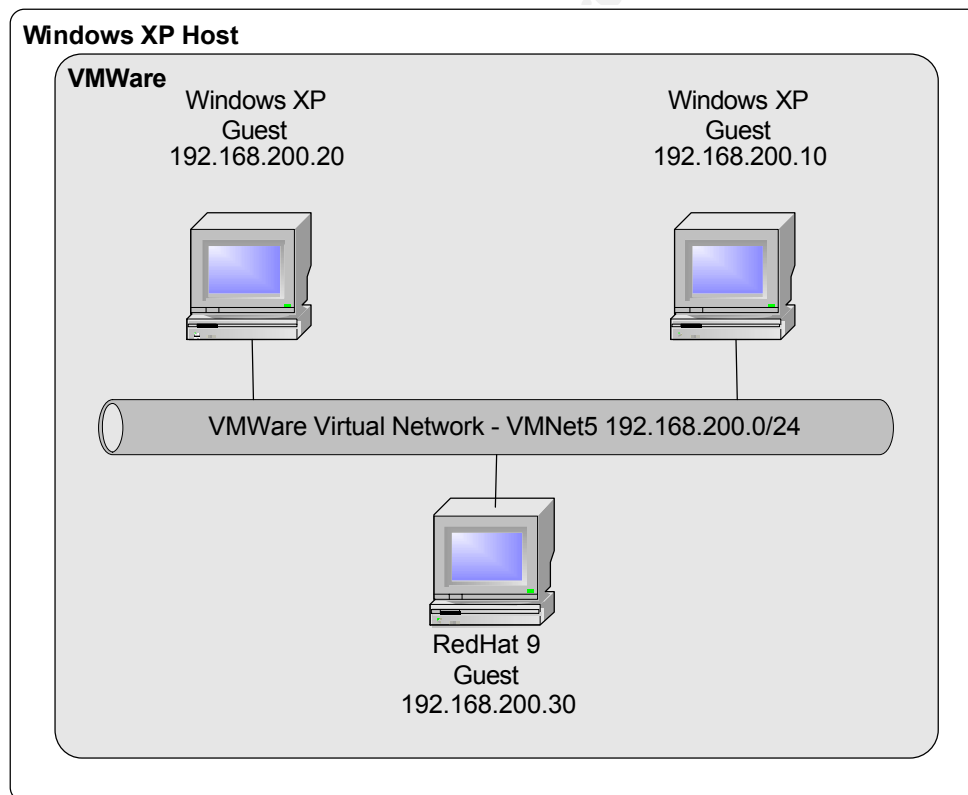


Figure 1: High Level Lab Architecture

The malware laboratory utilized for this analysis uses a virtual lab setup similar to the one utilized in class. The hardware is an IBM ThinkPad T40 laptop. The processor is a Pentium M running at 1500 MHz and containing 1 GB of RAM. The host operating system is Windows XP with Service Pack 1.

VMWare is utilized to setup the virtual lab inside the laptop. The VMWare version is 4.5.2 Build 8848, the most recent version available at the time of testing.

Three virtual machine (VM) images were prepared for the lab environment. The first and second virtual machines are nearly identical Windows XP with Service Pack 1 and all patches applied that were available as of December 17, 2004 except for XP Service Pack 2. A conscious decision was made to not apply SP2 since it is unclear how the networking changes in SP2 may impact the malware analysis process. The first virtual machine will be used as the launching point for the malware, and the second would be used if the malware uses a network based propagation vector.

The third virtual machine is a Red Hat 9 virtual machine. The malware we are analyzing only infects Windows based machines, so this virtual machine will not be infected by the malware. Rather, this machine is used to provide services to the lab. It will be used as a sniffer platform and may be used as an IRC server, DNS server, or other services as required in the analysis process.

The table below summarizes the virtual machines in the test environment.

Virtual Machine	Memory	Disk Size	OS	IP Address	Netmask
Windows 1	384 MB	6 GB	Windows XP SP2	192.168.200.10	/24
Windows 2	384 MB	6 GB	Windows XP SP2	192.168.200.20	/24
RedHat Linux	64 MB	2 GB	Red Hat 9.0	192.168.200.30	/24

Table 1: Virtual Machine Configurations

Please note that in the default lab configuration the default gateway and DNS server are not configured in any of the VMs. They can be configured if needed during testing.

Tools similar to those utilized in class were loaded onto the VMs. For a complete list of tools loaded in the environment please see Appendix A: Tools Utilized.

Once all the tools were loaded onto the VMs, snapshots were taken of each VM in the lab environment and zip archives of each of the VM directories were taken and burned to CD. This will permit the restoration to a pristine, uninfected version of the lab at any point, thus permitting repetition of the testing with minimal effort.

Several steps are taken to assure isolation of the lab environment. Firstly, the machine is unplugged from the LAN before any of the malware lab VMs are started. This assures that the malware cannot escape into the real world via the network while we are performing our tests.

Secondly a major departure was made from the testing environment used in class by isolating the testing environment from the host machine's interface by establishing a virtual network which the host does not participate in. The course material suggests using host-only networking. In a host-only network configuration, the VMs would each be attached to the built-in VMNet1 virtual network. On VMNet1, the host machine is assigned a virtual interface on the network to permit network communication between the VMs and the host operating system.

In this environment the VMs in the test environment are not configured to use host-only networking. Instead the VMs utilize virtual network VMNet5. The host does not participate in VMNet5, therefore the malware lab can be isolated from the host. This setup is done for each VM in the testing environment. These settings are accessed by opening the Virtual Machine Setting for the VM and then assign the network interface (VM->Settings, then click on NIC 1) to VMnet5 by clicking the button beside "Custom" and selecting "VMnet5" from the pull down menu.

The figure below shows the VMWare configuration screen.

© SANS Institute 2005, Author retains full rights.

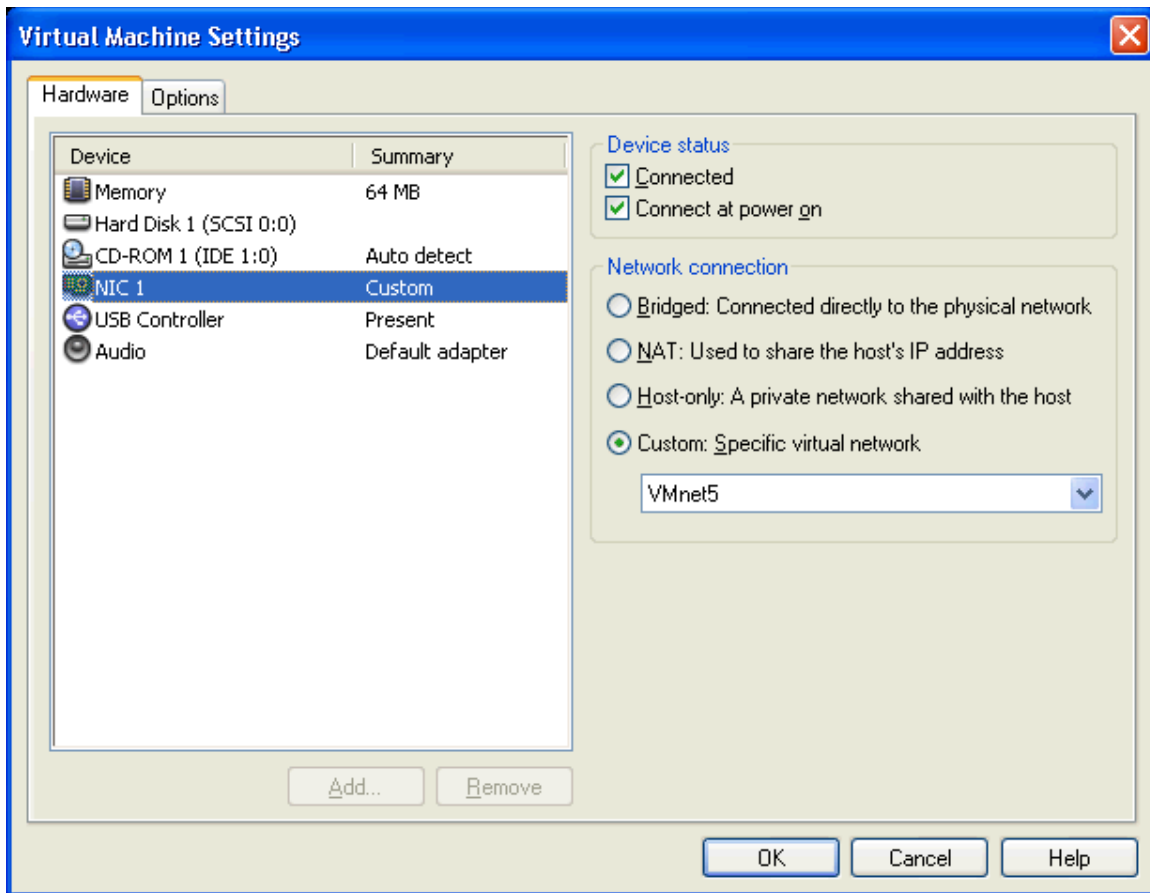


Figure 2: Virtual Machine Virtual Network Setup

By using a dedicated virtual network for the malware testing environment none of the VMs in the malware testing environment are able to communicate with the host operating system thus virtually eliminating the possibility of the host operating system being infected with malware.

Even so, precautions were taken with the host operating system. For anti-virus protection McAfee VirusScan Enterprise Edition version 7.1.0 is installed, and kept up to date with the latest anti-virus signatures. Also installed is McAfee Desktop Firewall version 8.0. The desktop firewall is configured to reject all traffic not initiated from the host. For anti-spyware protection Spybot Search and Destroy version 1.3 is installed and immunized with the most current signatures.

Another precaution was taken in the network local to the host machine. In the testing environment all the virtual machines utilize RFC 1918 addresses. If configuration mistakes were made which permitted the malware environment to communicate with the office LAN, the routers on our LAN utilize anti-spoofing ACLs which drop all traffic with a RFC1918 source or destination address. Assuming the malware doesn't spoof source addresses, this reduces the area of potential infection to the few machines on the local subnet.

One more step was performed before beginning the analysis. The VM to be infected with the malware was baselined so it would be easier to tell what effect the malware was having on the environment. The Active Ports tool was run locally to determine which ports were opened and by what applications. The results are in Appendix B – Active Ports Baseline. Also, a TCP and UDP nmap scan were run against the outside of the VM so an accurate external view of the VM was obtained before infecting it with the malware. These scans are in Appendix C Nmap Baselines. In addition Process Explorer was used to baseline the running processes on the VM. The output is in Appendix D Process Explorer Baseline.

Static Analysis

Properties of the Malware specimen

The first step in the malware analysis is to capture some statistics about the specimen itself. This is done to help identify the malware, and can be used later to see whether or not the malware changes at any stage during the analysis.

The specimen was provided to us from the courseware. It is named msrll.exe, and it is a Windows executable application. By opening a Windows command prompt window and changing to the directory containing the malware we can run the dir command to get some information about the specimen, such as the size. In this case the size of msrll.exe is 41,984 bytes. The command output is in the figure below:

```
02/01/2005 08:34 PM <DIR> .
02/01/2005 08:34 PM <DIR> ..
05/10/2004 04:29 PM          41,984 msrll.exe
          1 File(s)          41,984 bytes
          2 Dir(s)    1,587,212,288 bytes free
```

Figure 3: Malware Size

Running the md5sum command against the msrll.exe file provides us with a MD5 hash of the file. The hash gives us a benchmark against which to compare this version of the malware against others. Any change to the malware will result in a change to the MD5 hash, thus allowing us to detect the change. This also may be useful to help identify the version of the specimen when communicating with others about this malware specimen. The command to calculate the md5sum is simply md5sum msrll.exe. The figure below shows the command output

```
C:\Tools>md5sum msrll.exe
84acfe96a98590813413122c12c11aaa *msrll.exe
```

Figure 4: Malware md5sum

The '*' next to the filename tells us that the specimen is a binary file.

Embedded strings

Another source of potentially interesting information is the strings command. It will extract any ASCII strings from the file. The vast majority of this output is gibberish strings that are not likely to be of any value to us. However there are some nuggets that may be interesting. Please note that with the Windows version of strings the '-a' option must be used to get it to extract ASCII strings. By default it looks for Unicode strings.

The fact that there are not a lot of viewable strings suggests that the majority of the malware may be encrypted or packed in some manner.

The output of the strings command is too long to put into this document in its entirety, but Appendix E Summary of Interesting Strings Output, provides a summary.

The fact that there are DLL filenames in the strings output and the presence of the string "This program cannot be run in DOS mode." indicates that the malware is most likely a Windows based executable.

Also of interest in the strings output are a number of DLLs which the malware uses. A significant number of the DLLs referenced have to do with network communications. Although the exact purpose of these DLLs is not known at this point, it is likely that this malware will want to access the network.

A summary of the purpose of some of the referenced DLLs is below.

DLL	Description ¹
ws2_32.dll	Windows Sockets API used to manage network connections.
WS2HELP.dll	Functions used by the Windows Sockets API
shell32.dll	Windows Shell API functions, used when opening files.
wininet.dll	Internet related functions
rpcss.dll	Functions for distributed COM services
uxtheme.dll	Visual theme support
netapi32.dll	Windows NET API, used to do Microsoft networking
SETUPAPI.dll	Library used by installers and setup applications
Version.dll	Functions for Windows version checking
msvcrt.dll	Contains standard C library functions
user32.dll	Windows basic user interface API functions
kernel32.dll	Handles memory management, input/output operations and interrupts
advapi32.dll	Advanced API services library which supports many security and registry calls

Table 2: msrll.exe Referenced DLLs

¹Information for this section was taken from <http://www.liutilities.com/products/wintaskspro/dlllibrary/>

Also of interest are several function calls.

Function	Description ²
ExitProcess	Ends a process and all its threads
MessageBoxA	Provides ANSI formatting
wsprintfA	Print a string using the given format
GetProcAddress	Retrieves the address of an exported function or variable
GetModuleHandleA	Retrieves a module handle for a specified module
LoadLibraryA	Maps the specified executable module into the address space of the calling process
AdjustTokenPrivileges	Enables or disables privilege in an access token
itoa	Converts an integer to an ASCII string
getmainargs	Gets the arguments to the program
ShellExecuteA	Used to launch other Windows programs, open and print documents, and explore folders
DispatchMessageA	Sends a message to a window procedure
GetFileVersionInfoA	Retrieves version information for a specified file
InternetCloseHandle	Closes a file handle
WSAGetLastError	Returns error status for the last failed operation

Table 3: Functions in Strings Output

A significant number of these functions sound like functions most Windows programs would use. Perhaps the purpose will become clearer after further analysis.

The table below summarizes the properties of the malware specimen.

Filename	Type	Size	MD5 Hash
msrll.exe	Windows executable	41,984 bytes	84acfe96a98590813413122c12c11aaa

Table 4: Properties of the Malware Specimen

Behavioural Analysis

Now that we have learned a little about the malware from static analysis, we are ready to do a behavioural analysis. The basic technique will be to start a number of tools to watch the behaviour of the malware, and then execute the malware and analyze the output.

This will involve the following steps:

- 1) Use Regshot to capture a view of the registry. This is done by opening Regshot and clicking “1st shot” button and selecting “shot from the

² Information for this section was taken from <http://msdn.microsoft.com/library/>

menu. The figure below shows the Regshot interface with the “1st shot” button highlighted.

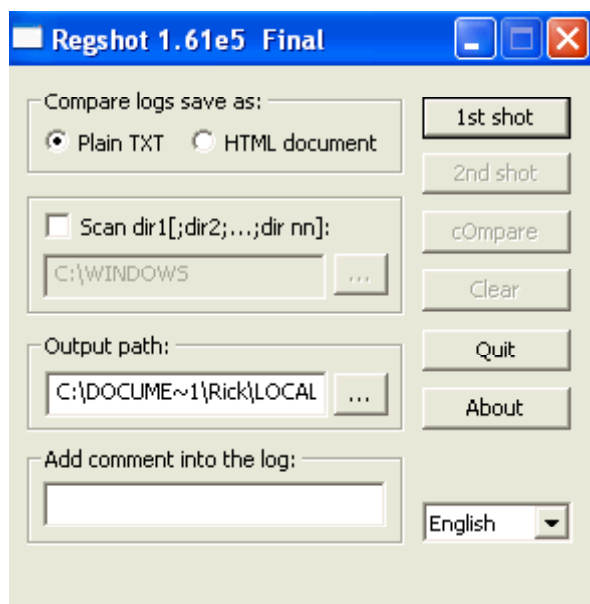


Figure 5: Regshot interface

- 2) Activate the Sysinternals tools to capture various aspects of the system while the malware is executing.
 - Filemon to monitor file accesses, creations, deletions, etc.
 - Regmon to monitor registry activity.
 - TDImon to monitor network/port activity.

All of the Sysinternals tools have the same interface. When the application is started it starts running. Stop it by clicking on the magnifying glass button, press the clear button (paper and erase icon). When you are ready to starting the malware, start the capture on all the tools by clicking on the magnifying glass icon. The figure below shows the Filemon interface as an example of all of the tools.

© SANS Institute

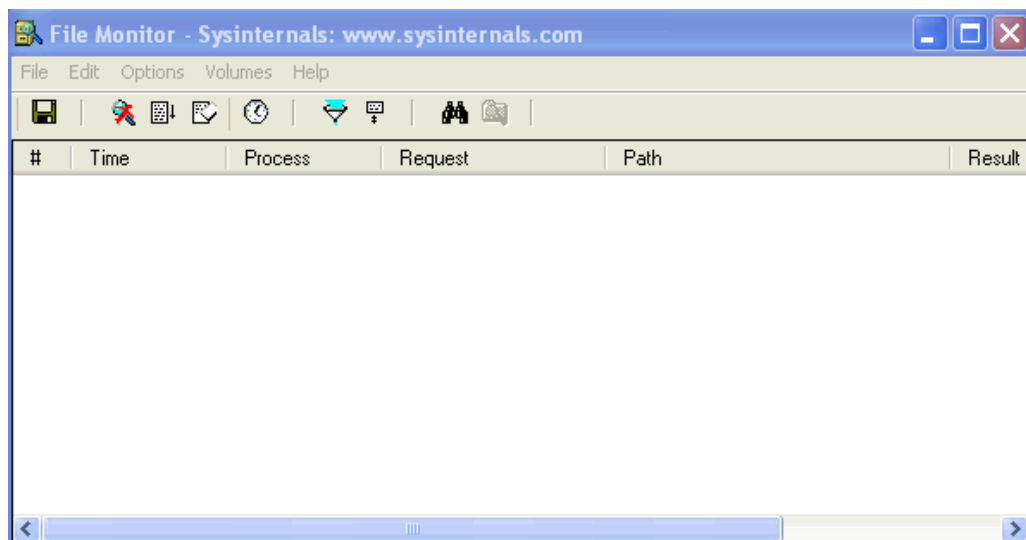


Figure 6: Sysinternals tools interface

- 3) Start ProcExp (Process Explorer) to monitor process activity.
- 4) Start up an external sniffer on the Linux VMWare image to capture the network traffic. In this case snort is being used. (`snort -vd | tee /tmp/malware.out`)
- 5) Execute the malware for approximately 30 seconds, then terminate the malware.
- 6) Stop the Sysinternals tools by clicking on the magnifying glass icon.
- 7) Take another Regshot to capture the view of the registry after the malware has executed. This is done by opening Regshot and clicking “2nd shot” button and selecting “shot from the menu. The figure below shows the Regshot interface with the “2nd shot” button highlighted.

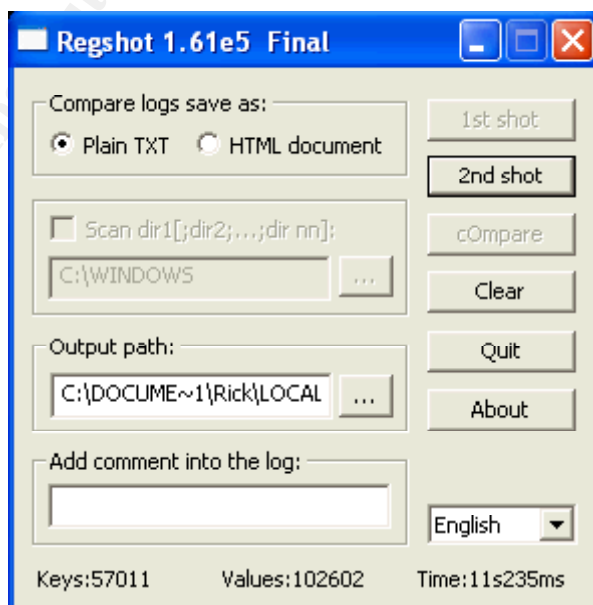


Figure 7: Regshot 2nd shot

- 8) Analyze the output. I prefer to save the Sysinternals tools output to a file (File -> Save As) and import it into Microsoft Excel. Since the Sysinternals tools output their files as tab delimited files, the import to Excel is relatively trivial. Just open the output file with Excel and the import wizard will automatically start. Tell the import wizard that the file is tab delimited and the import wizard will do the rest. Once in Excel the data is more easily sorted, manipulated and searched than in the Sysinternals tools.

It may be necessary to revert to the VMWare snapshot captured earlier and repeat these steps a number of times as we learn more about the malware.

First Iteration

On the first iteration of this exercise several interesting things were learned. First of all Process Explorer shows the creation of a process named msrll.exe.

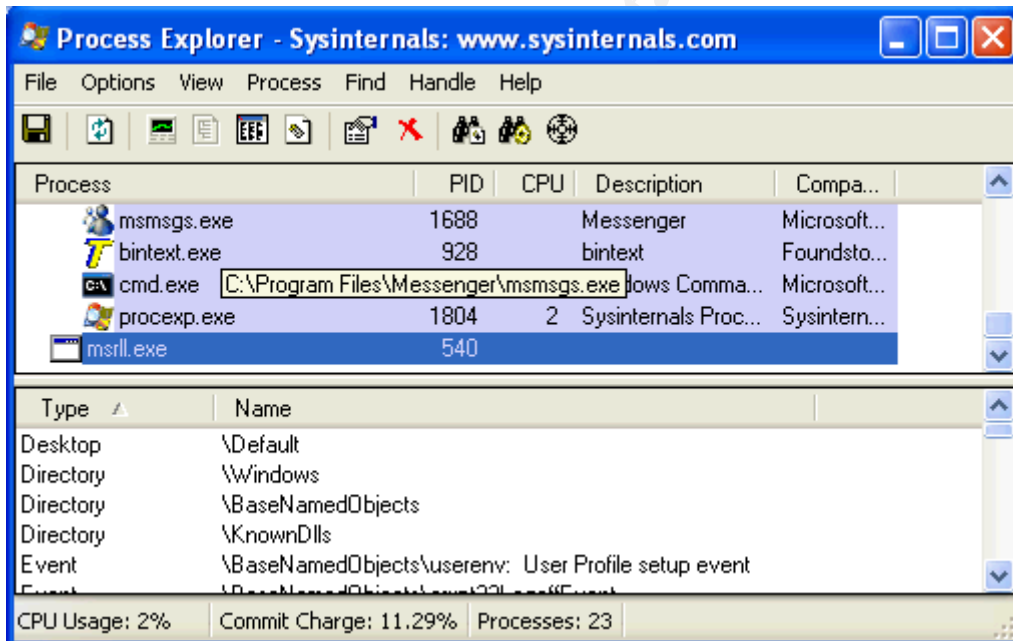


Figure 8: msrll.exe - Process Explorer

After the malware is terminated I saved the filemon, regmon, and tdimon output to files and imported it to Excel for easier manipulation.

The Filemon output shows the malware starting up and querying several DLLs. A number of the DLLs are the ones detected in the strings output. This output is quite lengthy, but an excerpt is shown below.

msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	FileNameInformation
msrll.exe:408	OPEN	C:\WINDOWS\Prefetch\MSRLL_EXE-1F3BB94A.pf	FILE NOT FOUND	Options: Open Access: All
msrll.exe:408	OPEN	C:\Documents and Settings\Rick\Desktop	SUCCESS	Options: Open Directory Access:
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\msrll.exe.Local	FILE NOT FOUND	Attributes: Error
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\ws2_32.dll	FILE NOT FOUND	Attributes: Error
msrll.exe:408	QUERY INFORMATION	C:\WINDOWS\System32\ws2_32.dll	SUCCESS	Attributes: A
msrll.exe:408	OPEN	C:\WINDOWS\System32\ws2_32.dll	SUCCESS	Options: Open Access: Execute
msrll.exe:408	CLOSE	C:\WINDOWS\System32\ws2_32.dll	SUCCESS	
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\WS2HELP.dll	FILE NOT FOUND	Attributes: Error
msrll.exe:408	QUERY INFORMATION	C:\WINDOWS\System32\WS2HELP.dll	SUCCESS	Attributes: A
msrll.exe:408	OPEN	C:\WINDOWS\System32\WS2HELP.dll	SUCCESS	Options: Open Access: Execute
msrll.exe:408	CLOSE	C:\WINDOWS\System32\WS2HELP.dll	SUCCESS	
msrll.exe:408	OPEN	C:\WINDOWS\system32\shell32.dll	SUCCESS	Options: Open Access: All
msrll.exe:408	QUERY INFORMATION	C:\WINDOWS\system32\shell32.dll	SUCCESS	Length: 8442368
msrll.exe:408	OPEN	C:\WINDOWS\system32\shell32.dll.124.Manifest	FILE NOT FOUND	Options: Open Access: All
msrll.exe:408	OPEN	C:\WINDOWS\system32\shell32.dll.124.Config	FILE NOT FOUND	Options: Open Access: All
msrll.exe:408	CLOSE	C:\WINDOWS\system32\shell32.dll	SUCCESS	

Figure 9: Filemon Output – First Iteration

Searching for CREATE in the msrll.exe portion of the Filemon output shows a directory and two new files created. First the malware creates a directory at C:\WINDOWS\System32\mfm then it creates and moves itself in that directory.

msrll.exe:408	CREATE	C:\WINDOWS\System32\mfm	SUCCESS	Options: Create Directory Access: All
msrll.exe:408	CLOSE	C:\WINDOWS\System32\mfm	SUCCESS	
msrll.exe:408	OPEN	C:\WINDOWS\System32\mfm	SUCCESS	Options: Open Directory Access: Traver
msrll.exe:408	CLOSE	C:\Documents and Settings\Rick\Desktop	SUCCESS	
msrll.exe:408	OPEN	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	Options: Open Sequential Access: All
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	Length: 41984
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	Attributes: A
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	Attributes: A
msrll.exe:408	CREATE	C:\WINDOWS\System32\mfm\msrll.exe	SUCCESS	Options: Overwritelf Sequential Access:
msrll.exe:408	OPEN	C:\WINDOWS\System32\mfm\	SUCCESS	Options: Open Access: 00000000
msrll.exe:408	CLOSE	C:\WINDOWS\System32\mfm\	SUCCESS	
msrll.exe:408	QUERY INFORMATION	C:\WINDOWS\System32\mfm\msrll.exe	SUCCESS	Attributes: A
msrll.exe:408	SET INFORMATION	C:\WINDOWS\System32\mfm\msrll.exe	SUCCESS	Length: 41984
msrll.exe:408	QUERY INFORMATION	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	Length: 41984
msrll.exe:408	WRITE	C:\WINDOWS\System32\mfm\msrll.exe	SUCCESS	Offset: 0 Length: 41984
msrll.exe:408	SET INFORMATION	C:\WINDOWS\System32\mfm\msrll.exe	SUCCESS	FileBasicInformation
msrll.exe:408	CLOSE	C:\Documents and Settings\Rick\Desktop\msrll.exe	SUCCESS	
msrll.exe:408	CLOSE	C:\WINDOWS\System32\mfm\msrll.exe	SUCCESS	

Figure 10: C:\Windows\system32\mfm creation and move of msrll.exe

Sure enough the malware has disappeared from the original directory in which it was located. A quick check on the size and an md5sum show that this file is identical to the original malware.

```

Directory of C:\WINDOWS\system32\mfm
01/19/2005  09:21 PM  <DIR>      .
01/19/2005  09:21 PM  <DIR>      ..
02/11/2005  09:59 AM                1,084 jtram.conf
05/10/2004  04:29 PM                41,984 msrll.exe
           2 File(s)                43,068 bytes
           2 Dir(s)            1,575,849,984 bytes free

C:\WINDOWS\system32\mfm>c:\tools\md5sum msrll.exe
34acfe96a98590813413122c12c11aaa *msrll.exe

```

Figure 11: C:\Windows\system32\mfm\msrll.exe size and md5sum

In the output above you can also see the presence of jtram.conf. The Filemon shows that the malware checked for the presence of C:\WINDOWS\System32\mfm\jtram.conf. Since the file did not exist on our system it created it. Filemon also shows a number of writes to that file.

OPEN	C:\WINDOWS\system32\mf\jtram.conf	FILE NOT FOUND	Options: Open Access: All
OPEN	C:\WINDOWS\system32\mf\jtram.conf	FILE NOT FOUND	Options: Open Access: All
CREATE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Options: Overwritelf Access: All
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 0 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 53 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 106 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 159 Length: 1
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 160 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 213 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 266 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 319 Length: 1
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 320 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 373 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 426 Length: 129
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 555 Length: 1
QUERY INFORM	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Attributes: A
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 556 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 609 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 662 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 715 Length: 1
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 716 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 769 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 822 Length: 77
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 899 Length: 1
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 900 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 953 Length: 53
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 1006 Length: 77
WRITE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	Offset: 1083 Length: 1
CLOSE	C:\WINDOWS\system32\mf\jtram.conf	SUCCESS	

Figure 12: Filemon - Creation of jtram.conf

The figure below shows that after creation jtram.conf has a size of 1084 bytes, and shows the computed md5sum.

```
C:\WINDOWS\system32\mf>c:\tools\md5sum jtram.conf
29c2de87c0679b9e1c2e80e44d7c7bd5 *jtram.conf

Directory of C:\WINDOWS\system32\mf

02/11/2005  10:12 AM                1,084 jtram.conf
             1 File(s)                  1,084 bytes
             0 Dir(s)  1,575,804,928 bytes free

C:\WINDOWS\system32\mf>c:\tools\md5sum jtram.conf
5f109c9d787bdd4dd8e20bb78710d997 *jtram.conf
```

Figure 12: C:\Windows\system32\mf\jtram.conf size and md5sum

Just hypothesizing, this file is probably either an output file, or a configuration file. In any case with the size and md5sum we can determine if this file changes in future iterations. A look at the content of this file is not very enlightening. jtram.conf appears to be some kind of binary file, probably compressed or encrypted somehow.

```
C:\WINDOWS\system32\mf>type jtram.conf
hf4RAELZHJhX18raHu143ir4IXZTmsKi4NUF14cEfhGvjQymww== 8wERAIyIUHYmXewOGaTnz8H1ayp
NCSgz yj1lAcoKwUHQe5H2dw== 9P4RAFFCm7u4hWf0pyUAsHg5tiH5GqMuBce6dr0ePo0hn1o7zQ==
RQIRAGUE/1PAaCpWU78eav+Jo0x7JYr8L1zCf7jtU4dweQG08g== UERAFm1DG4681C2h2jMq36KYeA
SSeAFZYq9qw5Ek1bd+3paWQ== 5v8RAcCQESi0m5NUc isX20e96gLcwUdYwxheU9k1m5vcq/nnKA==
fgARAKW9ScZSeNZde5ynsqbDснаFQQipj5FLECMc3Ns7ORqPg== gwERAF+EOxPS3iP1J8HRcfnmS0
OS0Bnr3Me7WGEpUfc1Kp9Uw== FABKADoBBX5MtvD2o1xP6wUPc1Bx1U6Cylq6eJ7dkt+JnlnGUSXjgf
5kcxnCt5Ff8fchfdeb9wrPJsYLWU3bRpt/TrrTN+2T2UJP57moJRFjxYFzxiP6ysPGb217g==
fgQRAOT2G1Ze8HeYtUZy1DuPZr2OeUorBFdXJctObJL6UXFYQg== KQARAPstOGNUUdBNgu1f7/7H5fm
3yh+rrnbbjGFmi/pa06PnSg== 4v8RAHPXa9ZF150cEzQq3yb1065CgXhWb0hkJUyAD0E0nzidEA==
JQERAFD93Izo6sUyAw2WMwrk1eGk+dmWJQP6QWkrDWxPA2Xtfw== c/4RAIF3IoeSnYTE3t0yfwJAUUm
doInvNCH+jA28UIC8cCFp3g== FvwjAP6KuWK6rwYSt7I1q+NgzMTd0e/TzwSUE21vn1+8YIbKv4iMEa
6L72w20nvIAGCUmOS6Lg==
lwERA08cSYDvX5zhQjMv1AInb0Uw4dUk5wEg1RhQbvd14T531A== 8gARAD+Nu313k/x371CLhRn02aq
mUSY1G09YabNccj2K0JjhdA== Uv8jAD8Dy7FXQI018j+Z0ns5BUFuQ9rRTKgPw0815IDDJoMKpoLU/Y
wUkLv yegZ/dmpDoMKfzQ==
```

Figure 13: jtram.conf file

The Regmon output shows the malware querying a number of keys in the HKLM hive, and the HKCU hive. The excerpt from the Regmon output is in the table below.

Action	Key	Value
OpenKey	HKLM\System\CurrentControlSet\Services	Key: 0xE1A13F08
CreateKey	HKLM\System\CurrentControlSet\Services\mf	Key: 0xE16C4B90
CloseKey	HKLM\System\CurrentControlSet\Services	Key: 0xE1A13F08
SetValue	HKLM\System\CurrentControlSet\Services\mf\T ype	0x120
SetValue	HKLM\System\CurrentControlSet\Services\mf\S tart	0x2
SetValue	HKLM\System\CurrentControlSet\Services\mf\E rrorControl	0x2
SetValue	HKLM\System\CurrentControlSet\Services\mf\I magePath	C:\WINDOWS\Sy stem32\mf\msrl l.exe
SetValue	HKLM\System\CurrentControlSet\Services\mf\D isplayName	Rll enhanced drive
CreateKey	HKLM\System\CurrentControlSet\Services\mf\S ecurity	Key: 0xE13FFA88
SetValue	HKLM\System\CurrentControlSet\Services\mf\S ecurity\Security	01 00 14 80 90 00 00 00 ...
CloseKey	HKLM\System\CurrentControlSet\Services\mf\S ecurity	Key: 0xE13FFA88
SetValue	HKLM\System\CurrentControlSet\Services\mf\O bjectName	LocalSystem

Table 6: msrll.exe Added as a Service

Research on support.microsoft.com³ indicates that the Services key and its subkeys are used to define a service and its parameters. The table below

³ <http://support.microsoft.com/kb/q103000/>

describes the purpose of the important subkeys.

Subkey	Value	Description
\Services\mfm\Type	0x120	Win32 program, shared process, user interaction
\Services\mfm\Start	0x2	Starts automatically on all startups
Services\mfm\ErrorControl	0x2	Severe - If service startup fails and the startup is not using the LastKnownGood control set, then switch to LastKnownGood. If already using LastKnownGood then continue
Services\mfm\ImagePath	C:\WINDOWS\System32\mfm\msrll.exe	The path to the executable for this service
Services\mfm\DisplayName	Rll enhanced drive	Name displayed for this service
Services\mfm\ObjectName	LocalSystem	The account the service will use to log on and run

Table 7: msrll.exe Service Subkey Descriptions

The critical item here is that the malware is configured to be started on all system startups.

From looking at the Tdimon output, it appears that IRP_MJ_CREATE is called whenever a port is open. Searching for IRP_MJ_CREATE in the Tdimon output for msrll.exe shows two ports being opened; TCP 2200, and TCP 113. The two tables below show the ports being opened and bound to.

msrll.exe:1444	IRP_MJ_CREATE	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	TDI_QUERY_INFORMATION	TCP:0.0.0.0:113	SUCCESS
msrll.exe:1444	IRP_MJ_CREATE	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj	SUCCESS
msrll.exe:1444	IRP_MJ_CREATE	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj	SUCCESS
msrll.exe:1444	IRP_MJ_CREATE	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:113	SUCCESS

Table 8: TDIMON - Port 113 Open

One of the common uses of TCP 113 is identd, a protocol which is used to

authenticate IRC connections.

msrll.exe:1444	IRP_MJ_CREATE	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	TDI_QUERY_INFORMATION	TCP:0.0.0.0:2200	SUCCESS
msrll.exe:1444	IRP_MJ_CREATE	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj	SUCCESS
msrll.exe:1444	IRP_MJ_CREATE	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj	SUCCESS
msrll.exe:1444	IRP_MJ_CREATE	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_ASSOCIATE_ADDRESS	TCP:Connection obj	SUCCESS
msrll.exe:1444	TDI_SET_EVENT_HANDLER	TCP:0.0.0.0:2200	SUCCESS

Table 9: TDIMON - Port 2200 Open

The snort output shows a DNS lookup for collective7.zxy0.com.

```
12/25-09-09:43:53.245995 192.168.200.10:1026 -> 192.168.200.1:53
UPD TTL:128 TOS:0x0 ID:629 IpLen:20 DgmLen:66
Len: 38
00 20 01 00 00 01 00 00 00 00 00 00 0B 63 6F 6C   . . . . .col
6C 65 63 74 69 76 65 37 04 7A 78 79 30 03 63 6F   lective7.zxy0.co
6D 00 00 01 00 01                                 m.....
```

Restarting msrll.exe permits us to attempt to connect to port 2200 to determine what sort of answer will be received, if any. Using netcat we get a prompt from the malware, but it does not respond to the input.

```
[root@localhost root]# nc 192.168.200.10 2200
#:hello
help
[root@localhost root]#
```

14: netcat output

This is also a good opportunity to verify that 113/tcp and 2200/tcp are both open. Using nmap (nmap -sV -p 113,2200 192.168.200.10) from the Linux VM confirms that both of those ports are indeed open. By adding the -sV flag to the nmap, it will attempt to determine the type and version of the applications bound to the ports. Nmap's best guess is that 113/tcp is auth, but it cannot recognize the application on port 2200/tcp.

```
[root@localhost root]# nmap -sV -p 113,2200 192.168.200.10

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2005-02-11 14:31 CST
Interesting ports on 192.168.200.10:
PORT      STATE SERVICE VERSION
113/tcp   open  auth?
2200/tcp  open  unknown
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at http://www.insecure.org/cgi-bin/serv
cefp-submit.cgi :
SF-Port2200-TCP:V=3.50%D=2/11%Time=420D1625%P=i386-redhat-linux-gnu%(NULL
SF:,2,"#:")%r(GenericLines,2,"#:")%r(GetRequest,2,"#:")%r(HTTPOptions,2,"#
SF:")%r(RTSPRequest,2,"#:")%r(RPCCheck,2,"#:")%r(DNSVersionBindReq,2,"#:"
SF:)%r(DNSStatusRequest,2,"#:")%r(Help,2,"#:")%r(SSLSessionReq,2,"#:")%r(S
SF:MBProgNeg,2,"#:")%r(X11Probe,2,"#:")%r(LPDString,2,"#:")%r(LDAPBindReq,
SF:2,"#:")%r(LANdesk-RC,2,"#:")%r(TerminalServer,2,"#:")%r(NCP,2,"#:")%r(N
SF:otesRPC,2,"#:")%r(WMSRequest,2,"#:")%r(oracle-tns,2,"#:");

Nmap run completed -- 1_IP address (1 host up) scanned in 100.857 seconds
```

Figure 15: nmap output

This is also a good time to double check whether the jtram.conf does change from execution to execution. As can be seen from the figure below, although the size doesn't change after the second execution of msrll.exe, the md5sum does. The most likely supposition for this behaviour is that each time the malware executes current state information is recorded in this file.

```
Directory of C:\WINDOWS\system32\mfm
02/11/2005  01:12 PM                1,084 jtram.conf
             1 File(s)                1,084 bytes
             0 Dir(s)           1,575,784,448 bytes free

C:\WINDOWS\system32\mfm>c:\tools\md5sum jtram.conf
cdf68b531cb58fefedc59a920035c5dc *jtram.conf
```

Figure 16: jtram.conf size and md5sum

Second Iteration

For the second iteration we attempted to change the behaviour of the malware by satisfying the DNS request for collective7.zxy0.com. Rather than deploying a DNS server in our malware lab, it is easier to add an entry to the end of the C:\Windows\system32\drivers\etc\hosts file. We can use any IP address in this file, but for the sake of possible future tests, we point the entry to the Linux server. This line looks like

```
192.168.200.30    collective7.zxy0.com
```

We confirm the host file entry by pinging collective7.zxy0.com.

```

C:\WINDOWS\system32\mfms>ping collective7.zxy0.com
Pinging collective7.zxy0.com [192.168.200.30] with 32 bytes of data:
Reply from 192.168.200.30: bytes=32 time<1ms TTL=64
Reply from 192.168.200.30: bytes=32 time<1ms TTL=64
Reply from 192.168.200.30: bytes=32 time<1ms TTL=64
Reply from 192.168.200.30: bytes=32 time=1ms TTL=64
Ping statistics for 192.168.200.30:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 1ms, Average = 0ms

```

Figure 17: ping collective7.zxy0.com

The snort output shows us that with the entry the malware attempts to connect to collective7.zxy0.com. It tries three times to connect to port 6667/TCP, then three times on port 9999/TCP, and three times on port 8080/TCP then cycles back to port 6667. 6667/TCP is a port that is commonly used by IRC. Since none of these ports are open on this system, each connection attempt results in a reset being sent back. The snort output of this exchange is in Appendix F 6667/9999/8080 Connection Attempts .

Third Iteration

For the third iteration I set up and started an IRC server on port 6667 on my Linux server. The software used for this is ircd-hybrid-6.1. This is the IRC server distributed on the course CD. The IRC server is started by using the following steps:

```

su - ircd          # switch user to the irc user
./ircd            # start the irc daemon

```

When the malware is started this time, a number of events occur. First of all in the snort log we see the TCP three-way handshake for a connection to the IRC server running on port 6667.

Secondly we see the IRC server send a message back to the malware telling it that it is going to send an AUTH request. Third we see the IRC server send several packets to 113/TCP, the ident port, and the malware answering. Next we see the malware communicate to the IRC server on port 6667/TCP using a NICK of "kCpWBoWmy" and the IRC server responding with a welcome message.

```

02/14-10:48:03.455133 192.168.200.10:1639 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7924 IpLen:20 DgmLen:116 DF
***AP*** Seq: 0xF78582C6 Ack: 0x7BB18C56 Win: 0xFAA1 TcpLen: 20
55 53 45 52 20 4B 72 4A 68 56 6F 46 4E 46 78 4F USER KrJhVoFNFx0
6F 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A 4D o localhost 0 :M
4F 6E 45 41 49 67 47 70 52 67 54 51 4A 45 4C 63 OnEAIgGpRgTQJELc
54 47 4A 51 67 43 4C 41 4E 58 0A 4E 49 43 4B 20 TGJQgCLANX. NICK
6B 43 70 57 42 6F 57 4D 79 47 74 0A kCpWBoWMyGt.

```

Figure 18: Malware Connects to IRC Server

The malware then joins a channel named “#mils”.

```
02/14-10:48:38.931427 192.168.200.10:1639 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7934 IpLen:20 DgmLen:53 DF
***AP*** Seq: 0xF7858325 Ack: 0x7BB192BE Win: 0xFAAA TcpLen: 20
4A 4F 49 4E 20 23 6D 69 6C 73 20 3A 0A          JOIN #mils :.
```

Figure 19: Malware joins IRC channel #mils

The complete snort output of this exchange is in Appendix G: Malware IRC Connection.

On the Linux VM I start an IRC client by typing the “irc” command. Once the IRC client has started I can join the channel by typing “/join #mils”. Using the “/names” command I can get a list of who is in the #mils channel. I see myself (root) and the malware (kCpWBoWMy).

```
*** - This is an IRC server. Authorized users only.
*** Mode change "+i" for user root by root
*** root (~root@127.0.0.1) has joined channel #mils
*** #mils 1104411362
Pub: #mils      root kCpWBoWMy
[1] 11:29 root (+i) on #mils (+nt) * type /help for help
```

Figure 20: Malware in the IRC channel #mils

I type a number of commands (login, who, status, hello, id, quit) and get no response from the malware. This suggests to me that it is likely that the malware expects some form of authentication sequence before responding to command input.

Killing and restarting the malware shows the malware logging into the channel using a different NICK each time. Most likely the NICK is randomly generated.

I wonder if now that the malware has joined an IRC channel if the behaviour of the port the malware opened (2200/TCP) has changed. However when I connect with netcat (nc 192.168.200.10 2200), I receive a prompt from the malware, but typing in the same commands as above does not elicit any response, and after two lines of commands the connection is terminated.

```
[root@localhost tmp]# nc 192.168.200.10 2200
#:login
help
[root@localhost tmp]#
```

Figure 21: Attempting to communicate with the malware on port 2200

I think we have reached a dead end as far as this line of investigation goes.

Perhaps we can find some clues that will permit interaction with the malware using code analysis.

Code Analysis

I started this phase of the testing with two goals. The primary goal being to determine the functionality of the malware by analyzing the command routines to determine what commands the malware responds to and the function of those commands. The secondary goal is determining the format and contents of the jtram.conf file. The malware appears to have two communication vectors, one through IRC, and the other through port 2200/tcp. In order to just isolate one of these to start, I am going to focus on the port 2200/tcp communications route for now. So the IRC server will not be started for now. If we have time we can come back to that channel later.

To begin the code analysis I loaded the malware into IDA Pro. IDA Pro is a disassembler, which will allow us to glean some information about the inner-workings of the malware. Unfortunately upon loading the malware into IDA Pro an IDA Pro warning message is received indicating that the file has been packed or modified in order to thwart analysis.

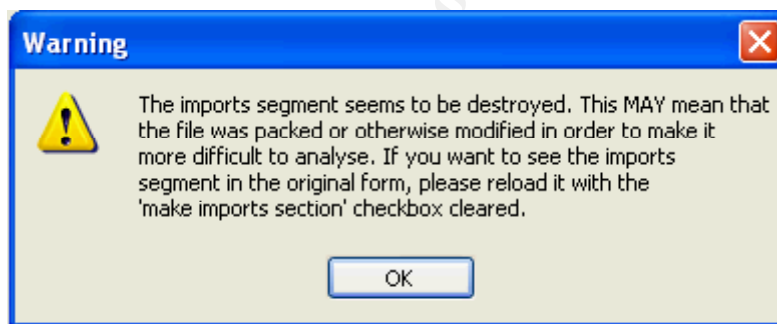


Figure 22: IDA Pro warning message

Continuing with the load, IDA Pro shows the majority of the code prefixed with .aspack.

```

IDA View-A
.aspack:0051D001
.aspack:0051D001 ; :::::::::::::::::::: S U B R O U T I N E ::::::::::::::::::::
.aspack:0051D001
.aspack:0051D001
.aspack:0051D001      public start
.aspack:0051D001 start      proc near
.aspack:0051D001      pusha
.aspack:0051D002      call     near ptr loc_51D007+3
.aspack:0051D007      loc_51D007:
.aspack:0051D007      jmp     near ptr 45AED4F7h ; COD
.aspack:0051D007 start      endp

```

Figure 23: .aspack prefix on imported code

A little Internet research reveals that the malware is compressed with ASPACK⁴. Some more research reveals a plethora of tools to unpack ASPACK⁵. The one I chose was AspackDie version 1.41⁶. AspackDie is a GUI based utility that uncompresses executables or DLLs compressed by Aspack. When started AspackDie opens a window asking which file to unpack. Once a file has been selected it runs for 10 or 15 seconds and then provides a success or failure dialog. If successful, the unpacked version is written to the same directory as the original binary but with a name of unpacked.exe.

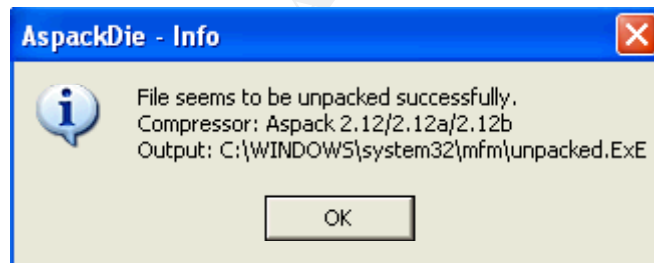


Figure 24: AspackDie success message

Once unpacked, and it is confirmed that the unpacked malware will load into IDA Pro properly, new vital statistics are captured for the malware. The unpacked version of the malware is nearly 1.12 MB instead of the original 41 KB. At this point a new MD5 checksum is also generated so we can clearly identify the unpacked version of this malware.

⁴ <http://www.aspack.com/aspack.html>

⁵ <http://protools.reverse-engineering.net/unpackers.htm>

⁶ <http://scifi.pages.at/yoda9k/files/AspackDie141.zip>

```

Directory of C:\WINDOWS\system32\mfm
01/14/2005  02:43 PM    <DIR>          .
01/14/2005  02:43 PM    <DIR>          ..
02/14/2005  01:41 PM             1,084 jtram.conf
05/10/2004  04:29 PM             41,984 msrll.exe
01/14/2005  02:39 PM             1,175,552 unpacked.ExE
              3 File(s)      1,218,620 bytes
              2 Dir(s)    1,565,077,504 bytes free

C:\WINDOWS\system32\mfm>c:\tools\md5sum unpacked.exe
d05c747e2158eb2b50643fee5c4ad338 *unpacked.exe

```

Figure 25: Unpacked malware size and md5sum

Now that the malware has been unpacked it shows much more string information. Analyzing the malware with BinText, we can see strings which look like error messages, and output strings that weren't visible before unpacking. The collective7.zxy0.com string is now visible, as is the channel name (#mils) and the configuration file name (jtram.conf). We can also see a number of strings that may be commands. The complete BinText output is in Appendix H BinText output for unpacked malware.

Potential Commands								
?!fif	?akick	?aop	?cd	?clone	?clone s	?con	?copy	?crash
?dcc	?dccs k	?del	?die	?dir	?dump	?echo	?exec	?fif
?free	?get	?hostnam e	?hush	?join	?jump	?kb	?kill	?killall
?killsk	?login	?ls	?md5 p	?mkdir	?move	?msg	?nick	?op
?part	?play	?ps	?pwd	?raw	?reboot	?rmdi r	?run	?say
?set	?si	?sklist	?ssl	?statu s	?sums	?uattr	?unset	?update
?uptim e	?wget	?smurf	?jolt	?udp	?syn			

Table 10: Potential Malware Commands

Unfortunately, when we type these strings into the IRC client, the malware does not respond, reaffirming that there is probably a password or some kind of hidden string to activate the malware communication.

More analysis of the BinText output reveals a couple of strings that look like MD5 hashes, possibly encrypted passwords.

Offset	Potential Password Strings
0040BDE0	\$1\$KZLPLKDF\$W8kl8Jr1X8DOHZsmlp9qq0
0040BE20	\$1\$KZLPLKDF\$55isA1ITvamR7bjAdBziX.

Table 11: Possible Password Strings

Unfortunately, running a dictionary attack utilizing The Argon⁷ password list and John the Ripper⁸ does not succeed in breaking the passwords.

Analyzing the Command Input via Port 2200

The general pattern of function calls for waiting on a port in Windows `bind()`, `listen()`, `accept()`, `recv()`⁹. It is likely that unless the malware author wrote his own input/output routines that the `recv()` system call is used to wait for input on port 2200.

Using IDA Pro's jump to function option (Jump -> Jump to function) I locate the `recv()` function. Double clicking on the result jumps the code window to the `recv()` function. From there right clicking on `recv` and selecting "Jump to xref to operand" will list all places in the code where `recv()` is called. The output reveals that `recv()` is only called from two places, `sub_40E66A` at offset `40E70D`, and in `sub_40D8CE` at offset `40DD22`.

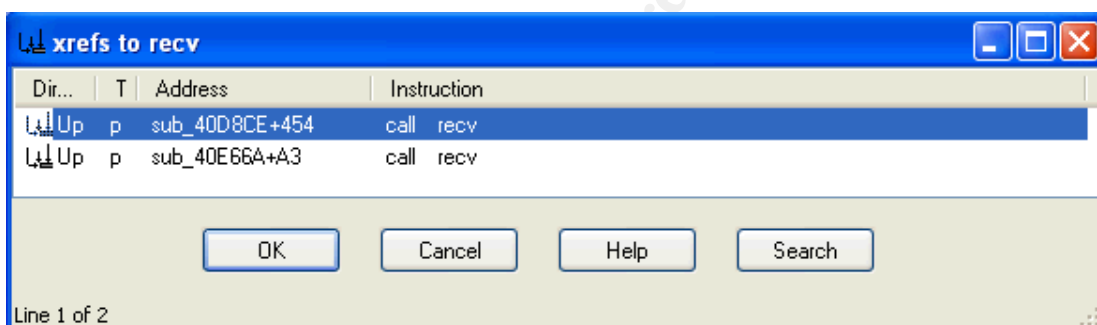


Figure 26: Xrefs to recv()

Using OllyDbg, a tracepoint is set immediately after the `recv()` calls at `40E712` and `40DD27`. In OllyDbg a breakpoint is set by right clicking over the address where you want to add the breakpoint and selecting "Breakpoint" and the type of breakpoint you want. OllyDbg supports a number of different types of breakpoints, but in this case we want the breakpoint to trigger anytime that section of the code is executed, so I selected "Breakpoint -> Toggle". To verify the breakpoints select "View -> Breakpoints".

⁷ <http://www.theargon.com/archives/wordlists/theargonlists/>

⁸ <http://www.openwall.com/john/>

⁹ <http://www.coding-zone.co.uk/cpp/articles/140101networkprogrammingm.shtml/>



Figure 27: After recv() breakpoints

Select “Debug -> Run” to start the malware executing inside OllyDbg. Using the Linux image, use netcat to connect to port 2200 (nc 192.168.200.10 2200). Once again the “#:” prompt is received from the malware. Enter “?login notpasswd” and press Enter. By using notpasswd as our password, it provides a recognizable string for easier detection inside the debugger. This input triggers the tracepoint on the recv() after 40E712. In OllyDbg, when a breakpoint is hit the box in the lower left corner of the display will turn yellow and contain the word “Paused”.

In order to short cut the analysis somewhat we can make some assumptions from standard coding practices. Generally, input routines only do input, they do not do any data validation or data processing. If this assumption is correct then the data validation routines will be one or more levels above this routine. From our earlier investigation we know that 40E712 is in sub_40E66A. Using the same techniques as before for checking the cross references of procedures in IDA Pro we see that sub_40E66A is only called from one place, in sub_40D8CE at offset 40DF9F.

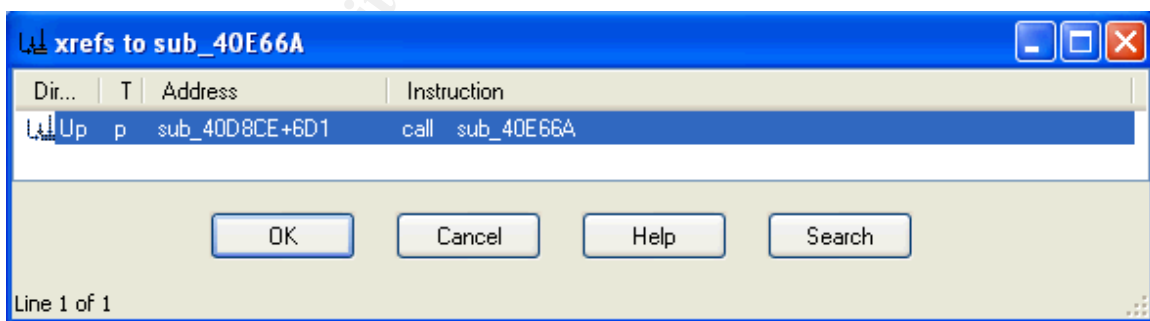


Figure 28: Calls to sub_40E66A

Placing a breakpoint after this point at 40DFA4, disabling the breakpoint at 40E712, restarting the malware, and using netcat to provide input, we can validate this assumption.

Sure enough the breakpoint at 40DFA4 is triggered on input. In OllyDbg’s registers window, the EDX register contains a pointer to the input string.

```

Registers (FPU)
EAX 00000001
ECX 0025B7D8
EDX 0041A1A8 ASCII "?login notpasswd"
EBX 004189A4 msrll.004189A4
ESP 0022CDF0
EBP 0022FE98
ESI 00000001
EDI 00002034
EIP 0040DFA4 msrll.0040DFA4

```

Figure 29: Input string pointer in EDX

What we are looking for after this point is the code where the input string is passed to validate the password. Toward this end we can walk through this code and determine the sequence of events. The methodology to be used is to utilize the step buttons in OllyDbg. OllyDbg contains three buttons which can be used to step through code. These buttons are all along the top above the CPU window. The first is “Step Into” (F7), it is used when the program counter is positioned on a procedure call to execute the next instruction inside the procedure call. The next is “Step Over” (F8), it is used to execute the next instruction in the current procedure. The last is “Execute till Return” (Ctrl+F9), it will trace over all statements in the current code until a return statement is encountered.

Using “Step Over” once the ADD ESP, 10 statement is executed. The current highlighted statement is then CMP EAX, 1. If we look at the EAX register, the content is 1, so this compare should succeed. Using “Step Over” once more, this statement is executed. The current highlighted statement is JE msrll.0040DEFC, since the compare was equal in the previous statement this jump should be executed. Looking at the box just below the CPU window, this Jump will be taken to 0040DEFC. Clicking “Step Over” again executes the jump.

0040DF98	. 50	PUSH EAX	Arg2
0040DF99	. FFB5 78CFFFFF	PUSH DWORD PTR SS:[EBP-3088]	Arg1
0040DF9F	. E8 C6060000	CALL msrll.0040E66A	msrll.0040E66A
0040DFA4	. 83C4 10	ADD ESP, 10	
0040DFA7	. 83F8 01	CMP EAX, 1	
0040DFAA	. ^0F84 4CFFFFFF	JE msrll.0040DEFC	
0040DFB0	. 83F8 FF	CMP EAX, -1	
0040DFB3	. v75 63 FF	JNZ SHORT msrll.0040E018	
0040DFB5	. 89F0	MOV EAX, ESI	
0040DFB7	. C1F0 0C	SHL EAX, 6	

Jump is taken
0040DEFC=msrll.0040DEFC

Figure 30: Jump Equal in input routine

Continuing the “Step Over” commands we eventually find some interesting code. At 40DF0F the code checks to make sure the first read character is not Null. The next statement is a JE (Jump Equal), it is not equal so the jump is not taken. My assumption is that this is a check for a Null string.

Another interesting bit of code is the RPNE SCAS BYTE PTR ES:[EDI] instruction at 40DF4F. RPNE is Repeat Not Equal¹⁰ when used with conjunction with SCAS it essentially searches for the first byte that matches the accumulator

(AL). In this case the AL=00. My assumption is that this is a string length calculation by detecting the null that terminates the string.

At 40DF5C the malware makes a call to 40BB6B. Using “Step Into” we can follow that call. One of the first things this routine does is a compare of the string length calculated with the RPNE to 1E (30 decimal). If the result is less than or equal to 1E the code jumps to 0040BB8F.

Soon after the code jumps to msrll.0040BBC9 (at 40BB91). At this point a TEST BYTE instruction is used to test the input string to see if it contains an ASCII 40 (A space). Our string does have a space, so it jumps to 40BCA6. Following this path, soon takes the code to a return statement, and then back to the recv() in the original input routine. It appears that the input failed a validation test. This means that the original assumption that the password needed to be prefixed with “?login” is incorrect. Let’s try again with different input. Instead of “?login notpasswd” let’s try with just the string “notpasswdnotpasswd”.

```
[root@localhost ~]$ nc 192.168.200.10 2200
#
notpasswdnotpasswd
```

Figure 31: “notpasswdnotpasswd” input to netcat

Following the same pattern, when the code reaches the TEST BYTE instruction at 40BBC9, the test fails and the code continues instead of jumping to 40BCA6. Continuing with the “Step Over” instructions, at 40BBDF msrll.00405872 is called with two parameters. The first is an ASCII string dcc.pass and the second being the input string of “notpasswdnotpasswd”.

```
0022CD00 0041E2B0 Arg1 = 0041E2B0 ASCII "notpasswdnotpasswd"
0022CD04 0040BB40 Arg2 = 0040BB40 ASCII "dcc.pass"
```

Figure 32: Stacked parameters for msrll.00405872

Stepping into this procedure at 40588B msrll.0040E7EB is called with “dcc.pass” as a parameter. However the second parameter is not the input string so we can skip over this call. At 40589E, msrll.0040D611 is called with the input string and one of the MD5 hashes discovered in the strings analysis. Perhaps this is a comparison routine with password encoding/decoding.

```
0022CD10 0041E2B0 ASCII "notpasswdnotpasswd"
0022CD14 0022CD20 ASCII "$1$KZLPLKdf$55isA1ITvamR7bjAdBz iX."
```

Figure 33: Stacked parameters for msrll.0040D611

Stepping into this procedure, at 0040D625, msrll.0040D587 is called, but the input string is not one of the parameters, so we will step over this procedure. At 40D644B msrll.0040D07E is called with a string of “KZLPLKdf”, and the input string as parameters.

¹⁰ <http://www.trotek.ec-lyon.fr/~muller/cours/8086/SCASB.html.en>

```

0022CCD0 0041E2B0 | Arg1 = 0041E2B0 ASCII "notpasswdnotpasswd"
0022CCD4 0022CCF0 | Arg2 = 0022CCF0 ASCII "KZLPLKdF"

```

Figure 34: Stacked parameters for msrll.0040D07E

Inside of this procedure, it appears that it is encoding the input string. My guess is that “KZLPLKdF” is a key which is being applied to the input string.

When msrll.00405872 returns the Registers window still has the original MD5 hash, but it has gained another string that also looks like an MD5 hash. It appears that this routine took the input string and generated a hash to compare to the password hash.

```

EDX 0022CD2C ASCII "55isA1ITvamR7bjAdBziX."
EBX 0022CD20 ASCII "%1$KZLPLKdF$55isA1ITvamR7bjAdBziX."

```

Figure 35: Hashes in registers

Because two MD5 hashes are compared, not two unencrypted strings, it will be nearly impossible to figure out the real password without reverse engineering the encryption routine. We need to find another way to bypass the password routine, perhaps by modifying the code.

At 40BBE7 the result of the msrll.00405872 is checked by performing a TEST EAX, EAX, essentially a check of the return code from the procedure. If this TEST succeeds, the code will jump to msrll.0040BC5A. That code is an error routine that indicates a bad password was entered. This instruction may be the key to bypassing the password validation routine, and accessing the command processing routines.

```

0040BBDF . E8 8E9CFFFF CALL msrll.00405872
0040BBE4 . 83C4 10     ADD ESP,10
0040BBE7 . 85C0       TEST EAX,EAX
0040BBE9 . 74 6F     JE SHORT msrll.0040BC5A

```

Figure 36: Test for valid password

We have a couple of options here. We can change the JE to a JNE which will reverse the behaviour of this code, or more easily we can just eliminate the JE opcode by replacing it with NOPs.

In OllyDbg right clicking over the JE statement will bring up the menu, select “Binary -> Fill with NOP’s” from the menu and the JE statement will be replaced with NOPs. Now this code will execute without performing the jump and allow us to bypass the password check.

```

0040BBDF . E8 8E9CFFFF CALL msrll.00405872
0040BBE4 . 83C4 10     ADD ESP,10
0040BBE7 . 85C0       TEST EAX,EAX
0040BBE9 90         NOP
0040BBEA 90         NOP
0040BBEB . 83EC 0C     SUB ESP,0C

```

Figure 37: JE replaced with NOPs

Now if we click the run button (or select Debug -> Run). The first thing that is obvious is that the behaviour of the connection has changed. The malware will now respond to commands, and on commands which it does not understand, it echos the command back along with the input string that we used as a password.

```
?status
service:N user:Rick inet connection:Y contype: Lan reboot privs:Y
help
(notpasswdnotpasswd) help
```

Figure 38: ?status - Response from the malware

We can utilize this parrot feature to test the validity of the commands in Table 7. By providing these commands to the malware if the command is not echoed back we know that it is valid. When tested all of the commands in Table 7 are valid. The output of the test is in Appendix I Commands Test. Some of the commands have obvious purposes. For example, ?reboot reboots the machine, and ?die kills the malware, ?hostname displays the hostname of the machine, ?uptime runs the uptime command, ?ls and ?dir list the current directory, ?pwd prints the current working directory, ?ps shows the process table, ?mkdir is used to create a directory, ?rmdir to remove directories, etc. Some commands appear to be for manipulation of an IRC session. ?set displays the malware's current parameters. ?sums generates checksums for files in the current directory. There is even a ?update command that appears to be for updating the malware. Also, it appears that the all of the commands work with a leading period instead of a question mark. Although this exercise confirms the purpose of some of the commands, it doesn't tell us what other commands may exist or the syntax of the non-obvious commands. Toward that end let's look at the way the ?status command is implemented.

The ?status command provides information about the connection including the user the process is running as. The string information from the ?status command output may be able to help us figure out where the rest of the commands are. Using the BinText output and searching for the static string "inet connection", we see that it is located at 405D40.

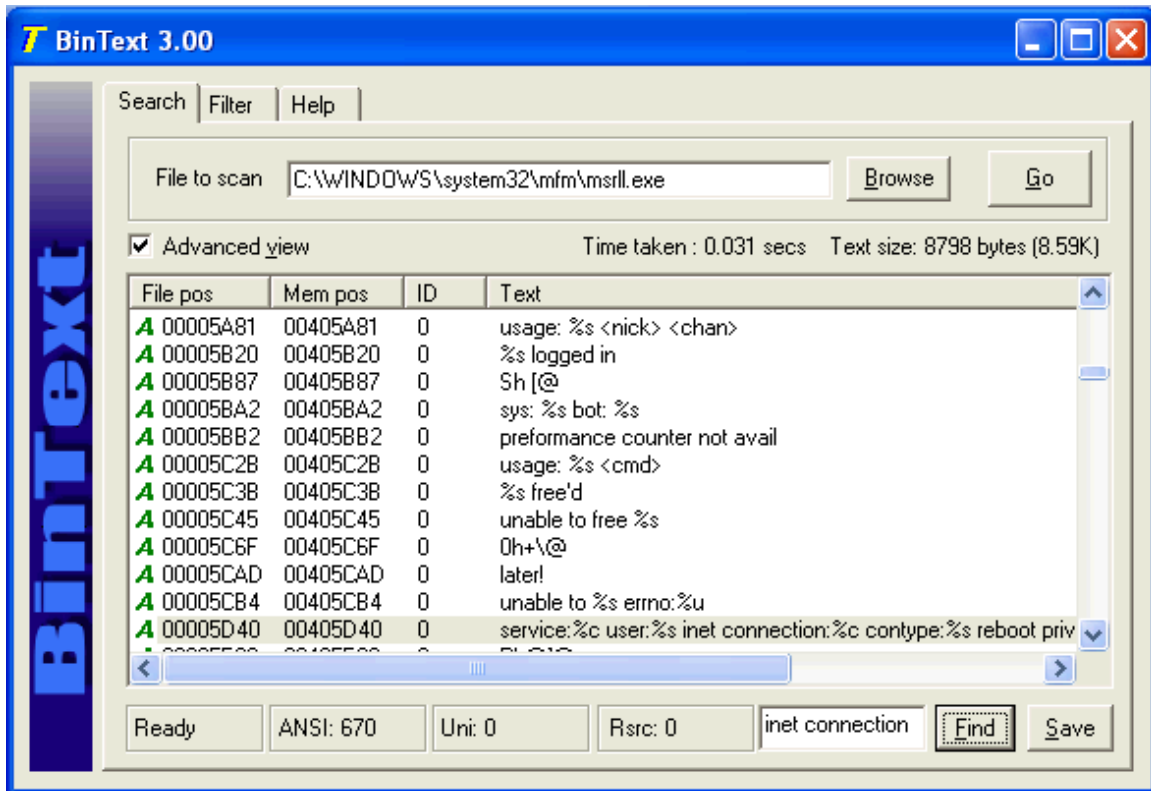


Figure 39: ?status output string

Using IDA Pro to find the xrefs to the string we find it is only referenced in one place, at 405E0A.

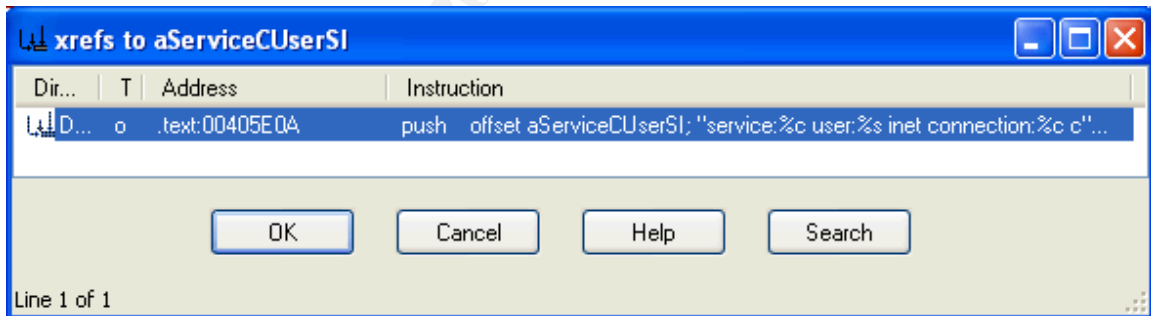


Figure 40: xrefs to ?status output string

Unfortunately this is where the wheels come off my cart. Despite a large amount of effort, I was not able to figure out how the command processing routines work, and therefore was not able to figure out all the commands the malware would respond to and their purposes.

Analyzing the jtram.conf File

The primary goal of this section is to figure out the contents of the jtram.conf file. Making the assumption that the jtram.conf string will need to be used to create the file, we can search the BinText output, and see that the string jtram.conf is

found in three places in the malware, at 4069EB, at 4099E0, and at 40C2B9. Utilizing the same methodology as before we can use IDA Pro to figure out where each of these strings is referenced, and analyze the code around these areas. The reference at 409D94 seems especially promising, since it appears to be part of a CreateFileA system call. CreateFileA is a system routine to create an ASCII file.

```

00409D7F . 83EC 04      SUB ESP,4
00409D82 . 6A 00        PUSH 0
00409D84 . 68 80000000  PUSH 80
00409D89 . 6A 02        PUSH 2
00409D8B . 6A 00        PUSH 0
00409D8D . 6A 00        PUSH 0
00409D8F . 68 00000040  PUSH 40000040
00409D94 . 68 E0994000  PUSH msrll.004099E0
00409D99 . E8 C2870000  CALL <JMP.&KERNEL32.CreateFileA>
00409DA4 . 8985 DCEFFFFF MOV DWORD PTR SS:[EBP-1024],EAX
00409DA7 . 83C4 04      ADD ESP,4
00409DA9 . 83F8 FF      CMP EAX,-1
00409DAB . 70F8 E4000000 JE msrll.00409E94

```

```

hTemplateFile = NULL
Attributes = NORMAL
Mode = CREATE_ALWAYS
pSecurity = NULL
ShareMode = 0
Access = GENERIC_WRITE
FileName = "jtram.conf"
CreateFileA

```

Figure 41: jtram.conf CreateFileA

Inserting a breakpoint at 409D94, removing the current jtram.conf and restarting the malware sure enough we hit the breakpoint. Executing a dir command in the C:\Windows\System32\mf\ dir command we see that the file has not yet been created, however if we “Step Over” until after the CreateFileA system call and repeat the dir command, we see a zero length jtram.conf file has been created.

```

Directory of C:\WINDOWS\system32\mf\
02/22/2005  07:35 PM    <DIR>          .
02/22/2005  07:35 PM    <DIR>          ..
02/22/2005  07:35 PM                0 jtram.conf

```

Figure 42: Zero length jtram.conf file

Now we know where the file was created, now to just figure out where it is written, and what is written to it. Looking down further in this code there is an obvious call to WriteFile at 409E1E. Assuming this is where the data is written to the file, and already knowing that the data that goes into the jtram.conf file is encoded in some manner, it is probable that an encoding routine of some sort must happen between where the file is created and where it is written.

```

00409DEB . 83C4 08      ADD ESP,8
00409DEE . 68 129A4000  PUSH msrll.00409A12
00409DF3 . 56           PUSH ESI
00409DF4 . E8 87830000  CALL <JMP.&msvcrt.strcat>
00409DF9 . C70424 000000 MOV DWORD PTR SS:[ESP],0
00409E00 . 8D85 E4FFFFFF LEA EAX,DWORD PTR SS:[EBP-101C]
00409E06 . 50           PUSH EAX
00409E07 . 89F7        MOV EDI,ESI
00409E09 . FC          CLD
00409E0A . B0 00        MOV AL,0
00409E0C . B9 FFFFFFFF  MOV ECX,-1
00409E11 . F2:AE       REPNE SCAS BYTE PTR ES:[EDI]
00409E13 . F7D1        NOT ECX
00409E15 . 49          DEC ECX
00409E16 . 51           PUSH ECX
00409E17 . 56           PUSH ESI
00409E18 . FFB5 DCEFFFFF PUSH DWORD PTR SS:[EBP-1024]
00409E1E . E8 9D870000  CALL <JMP.&KERNEL32.WriteFile>
00409E23 . 83C4 0C      ADD ESP,0C

```

```

[ src = " "
  dest
  strcat
  pBytesWritten
  nBytesToWrite
  Buffer
  hFile
  WriteFile

```

Figure 43: jtram.conf WriteFile

Between these two points is not a great amount of code, and only one procedure call, to `msrll.0040B2B0`. Using “Step Over” to step down to this code, we see that the an ASCII string which could be an encryption string is pushed onto the stack and the first time the procedure is executed an ASCII string of “set “ is passed as one of the parameters.

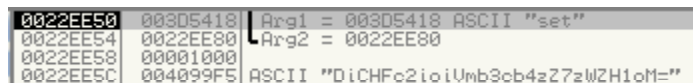


Figure 44: msrll.0040B2B0 stack parameters

Inserting a breakpoint at `409DE6` and iterating through this routine permits capturing these parameters off the stack and thus capturing the strings which are encoded and written to `jtram.conf`. The table below shows the captured strings.

set bot.port 2200
set irc.quit 003D46D0
set servers collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:8080
set irc.chan #mils
set pass \$1\$KZLPLKDF\$W8kl8Jr1X8D)HZsmlp9qq0
set dcc.pass \$1\$KZLPLKDF\$55isA1ITvamR7bjAdBziX.

Table 12: jtram.conf contents

Analysis Wrap-Up

Based on the analysis above, I believe this code to be a Trojan listener. It is clear that it has two potential ways of being controlled. The first being the IRC server on `collective7.zxy0.com`, channel `#mils`, and the second being directly via TCP port 2200.

From the strings and code analysis it appears that the Trojan has a number of capabilities. The presence of commands like `?smurf` and `?syn` indicate the presence of DOS capabilities. Also, numerous commands are present which permit remote control of various facets of the local machine including the ability to create, delete, move around in the directory structure, and list files in directories (`?mkdir`, `?rmdir`, `?dir`, `?ls`, `?pwd`, `?cd`), move and copy files (`?move`, `?copy`), view various system attributes (`?ps`, `?uptime`, `?si`), start and kill processes (`?run`, `?kill`, `?killall`), interact with the user (`echo`), and just generally cause havoc (`?crash`, `?reboot`). The malware also appears to have the ability to interact via IRC commands

This malware could get into the network via a number of methods, including email or file transfer. The best methodology for preventing the propagation of this malware via email is to block or quarantine executables at the email gateway. To detect and potentially block this malware via other file transfer

methods, an IPS/IDS could be deployed with the proper signatures to detect this malware.

If this malware is on your network already it can be detected via a port scan for port 2200/TCP, or by analyzing network traffic for communication with collective7.zxy0.com.

The ability of this malware to communicate with the Internet could be eliminated by not permitting unnecessary ports out of your network. In this case specifically 6667/TCP, 9999/TCP, and 8080/TCP should be explicitly blocked. As a general rule it is best practice to not permit externally initiated traffic into your network (i.e. block all inbound), this would block the ability for an external party to communicate with the malware. If externally initiated traffic is required for some reason, for example to an Internet facing server then explicitly blocking port 2200/TCP inbound would eliminate the communication mechanism.

© SANS Institute 2005, Author retains full rights.

References

Skoudis, Ed, Lenny Zeltser. Malware Fighting Malicious Code. Upper Saddle River, New Jersey: Prentice Hall, 2004.

Zeltser, Lenny. Reverse Engineering Malware: Tools and Techniques Hands-On, SANS Institute: 2004

Uniblue Systems LTD. WinTasks DLL Library. January 7, 2005.
<<http://www.liutilities.com/products/wintaskspro/dlllibrary/>>

Microsoft Corporation. MSDN Library. January 10, 2005.
<<http://msdn.microsoft.com/library/>>

ASPACK Software. ASPACK Software. January 13, 2005.
<<http://www.aspack.com/aspack.htm>>

Protools – Unpackers. Programmers Tools [Protools.owns.it]. January 13, 2005.
<<http://protools.reverse-engineering.net/unpackers.htm>>

y0da. AspackDie 1.41. January 13, 2005.
<<http://scifi.pages.at/yoda9k/proggies.htm>>

The A.R.G.O.N.. theargonlistver1.zip. January 10, 2005.
<<http://www.theargon.com/archives/wordlists/theargonlists/>>

Openwall Project. John the Ripper password cracker. January 10, 2005.
<<http://www.openwall.com/john/>>

Coding-Zone.co.uk. “Using Internet Sockets”. January 15, 2005.
<http://www.coding-zone.co.uk/cpp/articles/140101networkprogrammingm.shtml>

© SANS Institute. Author retains full rights.

Appendix A: Tools Utilized

Windows Tools

Tool Name	Version	Description	Location
md5sum.exe	2.0	Computes an MD5 hash of the input file	http://downloads.activestate.com/contrib/md5sum/Windows/md5sum.exe
strings.exe	2.1	Analyzes the input file and outputs ascii strings detected in the file	http://www.sysinternals.com/
filemon	6.07	File system monitor	http://www.sysinternals.com/
regmon	6.06	Registry monitor	http://www.sysinternals.com/
tdimon	1.0	Network port monitor	http://www.sysinternals.com/
Ollydbg	1.10	Debugger	http://home.t-online.de/home/Ollydbg/viewer.htm
Ida Pro	4.6.0.785	Disassembler with debugging features	http://www.datarescue.com/
bintext	3.00	A GUI based strings program	http://www.foundstone.com/
regshot	1.61e5	Takes snapshots of the registry and compares them.	http://www.sysinternals.com/
Active Ports	1.4	Shows open TCP and UDP ports	http://www.ntutility.com/
aspackdie	1.41	Unpacker for ASPACK	http://scifi.pages.at/yoda9k/projects/aspackdie.htm
Process Explorer	8.52	An enhanced task manager program.	http://www.sysinternals.com/

Unix Tools

Tool Name	Version	Description	Location
snort	2.0.4 Build 96	Packet sniffer with IDS capabilities	http://www.snort.org/dl/
nmap	3.50	Freeware port scanner. Used for TCP and UDP scans for baselining the VMWare images.	http://www.insecure.org/
Ircd	6.1	Internet Relay Chat (IRC) server	http://sourceforge.net/projects/ircd-hybrid/

Appendix B – Active Ports Baseline

This output is generated by running Active Ports and doing a File -> Save As and saving as type “Text (Tab Delimited) (*.txt)”

Process	PID	Local IP	Local Port	State	Proto	Path
System 4	192.168.200.10	138	LISTENUDP			
System 4	192.168.200.10	137	LISTENUDP			
System 4	0.0.0.0	445	LISTENUDP			
System 4	192.168.200.10	139	LISTENTCP			
System 4	0.0.0.0	445	LISTENTCP			
lsass.exe	668	0.0.0.0	500	LISTENUDP	C:\WINDOWS\system32\lsass.exe	
svchost.exe	836	0.0.0.0	135	LISTENTCP	C:\WINDOWS\system32\svchost.exe	
svchost.exe	936	192.168.200.10	123	LISTENUDP	C:\WINDOWS\System32\svchost.exe	
svchost.exe	936	0.0.0.0	1055	LISTENUDP	C:\WINDOWS\System32\svchost.exe	
svchost.exe	936	0.0.0.0	1025	LISTENTCP	C:\WINDOWS\System32\svchost.exe	
svchost.exe	1160	0.0.0.0	1026	LISTENUDP	C:\WINDOWS\System32\svchost.exe	
svchost.exe	1224	192.168.200.10	1900	LISTENUDP	C:\WINDOWS\System32\svchost.exe	
svchost.exe	1224	0.0.0.0	5000	LISTENTCP	C:\WINDOWS\System32\svchost.exe	
msmsgs.exe	1596	192.168.200.10	36982	LISTENUDP	C:\Program Files\Messenger\msmsgs.exe	
msmsgs.exe	1596	192.168.200.10	8147	LISTENUDP	C:\Program Files\Messenger\msmsgs.exe	
msmsgs.exe	1596	0.0.0.0	1027	LISTENUDP	C:\Program Files\Messenger\msmsgs.exe	
msmsgs.exe	1596	192.168.200.10	13087	LISTENTCP	C:\Program Files\Messenger\msmsgs.exe	

Appendix C Nmap Baselines

TCP Baseline

```
[root] nmap -p 1-65535 192.168.200.10
```

Starting nmap 3.50 (<http://www.insecure.org/nmap>) at 2004-12-17 14:13 CST

Interesting ports on 192.168.200.10

(The 65529 ports scanned but not shown below are in state: closed)

PORT	STATE	SERVICE
135/tcp	open	msrpc
139/tcp	open	netbios-ssn
445/tcp	open	microsoft-ds
1025/tcp	open	NFS-or-IIS
5000/tcp	open	UPnP
13087/tcp	open	unknown

Nmap run completed – 1 IP address (1 host up) scanned in 88.382 seconds.

UDP Baseline

```
[root] nmap -sU -p 1-65535 192.168.200.10
```

Starting nmap 3.50 (<http://www.insecure.org/nmap>) at 2004-12-17 14:22 CST

Interesting ports on 192.168.200.10

(The 65524 ports scanned but not shown below are in state: closed)

PORT	STATE	SERVICE
123/udp	open	ntp
137/udp	open	netbios-ns
138/udp	open	netbios-dgm
445/udp	open	Microsoft-ds
500/udp	open	isakmp
1026/udp	open	unknown
1027/udp	open	unknown
1055/udp	open	unknown
1419/udp	open	Timbuktu-srv3
1900/udp	open	UPnP
8147/udp	open	unknown

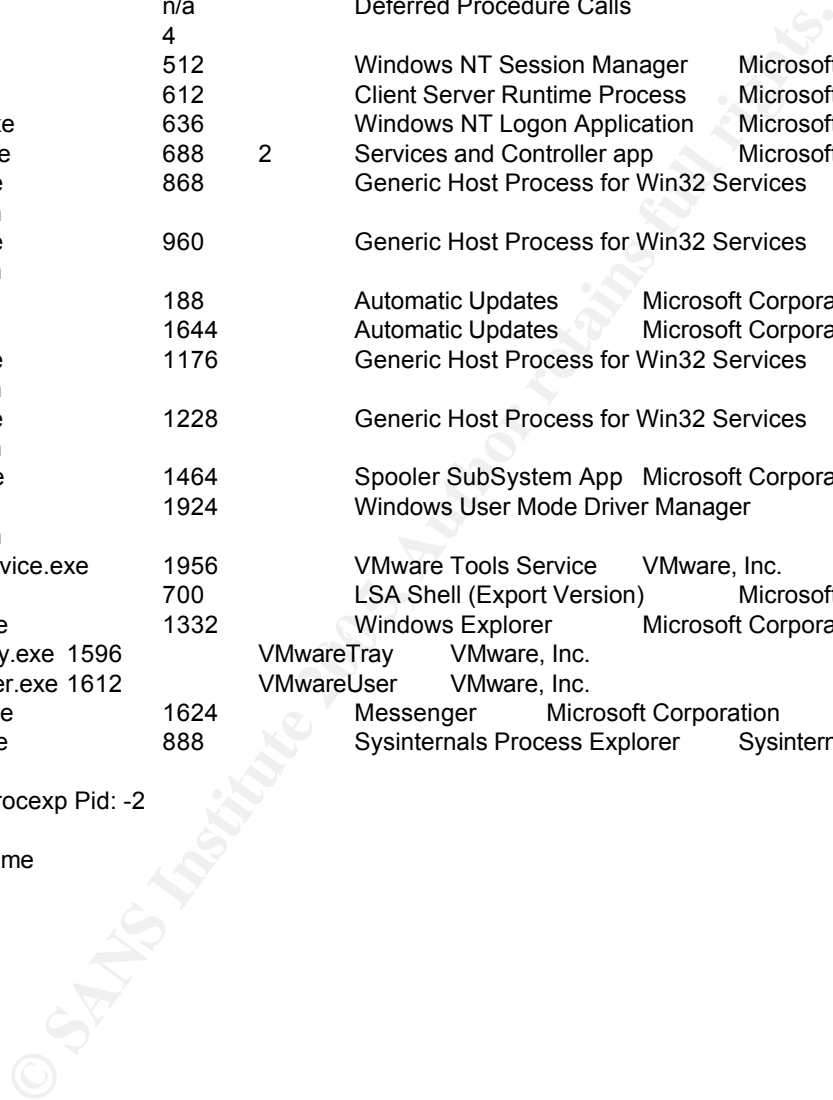
Appendix D Process Explorer Baseline

This output is generated by running Process Explorer and doing a File -> Save As and saving as type "Process Explorer Data (*.txt)"

Process	PID	CPU	Description	Company Name
System Idle Process	0	98		
Interrupts	n/a		Hardware Interrupts	
DPCs	n/a		Deferred Procedure Calls	
System	4			
smss.exe	512		Windows NT Session Manager	Microsoft Corporation
csrss.exe	612		Client Server Runtime Process	Microsoft Corporation
winlogon.exe	636		Windows NT Logon Application	Microsoft Corporation
services.exe	688	2	Services and Controller app	Microsoft Corporation
svchost.exe	868		Generic Host Process for Win32 Services	Microsoft Corporation
svchost.exe	960		Generic Host Process for Win32 Services	Microsoft Corporation
wuauclt.exe	188		Automatic Updates	Microsoft Corporation
wuauclt.exe	1644		Automatic Updates	Microsoft Corporation
svchost.exe	1176		Generic Host Process for Win32 Services	Microsoft Corporation
svchost.exe	1228		Generic Host Process for Win32 Services	Microsoft Corporation
spoolsv.exe	1464		Spooler SubSystem App	Microsoft Corporation
wdfmgr.exe	1924		Windows User Mode Driver Manager	Microsoft Corporation
VMwareService.exe	1956		VMware Tools Service	VMware, Inc.
lsass.exe	700		LSA Shell (Export Version)	Microsoft Corporation
explorer.exe	1332		Windows Explorer	Microsoft Corporation
VMwareTray.exe	1596		VMwareTray	VMware, Inc.
VMwareUser.exe	1612		VMwareUser	VMware, Inc.
msmsgs.exe	1624		Messenger	Microsoft Corporation
procexp.exe	888		Sysinternals Process Explorer	Sysinternals

Process: Procexp Pid: -2

Type Name



Appendix E Summary of Interesting Strings Output

```
C:\Tools> strings -a msrll.exe
Strings v2.1
Copyright (C) 1999-2003 Mark Russinovich
Systems Internals - www.sysinternals.com
```

```
!This program cannot be run in DOS mode.
```

```
.text
```

```
.data
```

```
.bss
```

```
.idata
```

```
.aspack
```

```
.adata
```

```
.
```

```
kernel32.dll
```

```
ExitProcess
```

```
user32.dll
```

```
MessageBoxA
```

```
wsprintfA
```

```
LOADER ERROR
```

```
The procedure entry point %s could not be located in the dynamic link library %s
```

```
The ordinal %u could not be located in the dynamic link library %s
```

```
.
```

```
kernel32.dll
```

```
GetProcAddress
```

```
GetModuleHandleA
```

```
LoadLibraryA
```

```
advapi32.dll
```

```
msvcrt.dll
```

```
msvcrt.dll
```

```
shell32.dll
```

```
user32.dll
```

```
version.dll
```

```
wininet.dll
```

```
ws2_32.dll
```

```
AdjustTokenPrivileges
```

```
_itoa
```

```
__getmainargs
```

```
ShellExecuteA
```

```
DispatchMessageA
```

```
GetFileVersionInfoA
```

```
InternetCloseHandle
```

```
WSAGetLastError
```

Appendix F 6667/9999/8080 Connection Attempts

02/11-15:19:28.248682 192.168.200.10:1535 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7250 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xAB07D84F Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

====+

02/11-15:19:28.252768 192.168.200.30:6667 -> 192.168.200.10:1535
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAB07D850 Win: 0x0 TcpLen: 20

====+

02/11-15:19:28.932056 192.168.200.10:1535 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7251 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xAB07D84F Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

====+

02/11-15:19:28.932072 192.168.200.30:6667 -> 192.168.200.10:1535
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAB07D850 Win: 0x0 TcpLen: 20

====+

02/11-15:19:29.624960 192.168.200.10:1535 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7252 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xAB07D84F Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

====+

02/11-15:19:29.624990 192.168.200.30:6667 -> 192.168.200.10:1535
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAB07D850 Win: 0x0 TcpLen: 20

====+

02/11-15:20:02.653545 192.168.200.10:1536 -> 192.168.200.30:9999
TCP TTL:128 TOS:0x0 ID:7253 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xAB74B816 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

====+

02/11-15:20:02.653574 192.168.200.30:9999 -> 192.168.200.10:1536
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAB74B817 Win: 0x0 TcpLen: 20

====+

02/11-15:20:03.115441 192.168.200.10:1536 -> 192.168.200.30:9999
TCP TTL:128 TOS:0x0 ID:7254 IpLen:20 DgmLen:48 DF

*****S* Seq: 0xAB74B816 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:20:03.115445 192.168.200.30:9999 -> 192.168.200.10:1536
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAB74B817 Win: 0x0 TcpLen: 20

=====
=====

02/11-15:20:03.675436 192.168.200.10:1536 -> 192.168.200.30:9999
TCP TTL:128 TOS:0x0 ID:7255 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xAB74B816 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:20:03.675440 192.168.200.30:9999 -> 192.168.200.10:1536
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAB74B817 Win: 0x0 TcpLen: 20

=====
=====

02/11-15:20:36.384999 192.168.200.10:1537 -> 192.168.200.30:8080
TCP TTL:128 TOS:0x0 ID:7256 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xABE1A6B8 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:20:36.385003 192.168.200.30:8080 -> 192.168.200.10:1537
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xABE1A6B9 Win: 0x0 TcpLen: 20

=====
=====

02/11-15:20:36.986899 192.168.200.10:1537 -> 192.168.200.30:8080
TCP TTL:128 TOS:0x0 ID:7257 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xABE1A6B8 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:20:36.986922 192.168.200.30:8080 -> 192.168.200.10:1537
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xABE1A6B9 Win: 0x0 TcpLen: 20

=====
=====

02/11-15:20:37.450858 192.168.200.10:1537 -> 192.168.200.30:8080
TCP TTL:128 TOS:0x0 ID:7258 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xABE1A6B8 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:20:37.450882 192.168.200.30:8080 -> 192.168.200.10:1537

TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xABE1A6B9 Win: 0x0 TcpLen: 20

=====
=====

02/11-15:21:44.311903 192.168.200.10:1538 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7259 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xACBA803E Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:21:44.314914 192.168.200.30:6667 -> 192.168.200.10:1538
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xACBA803F Win: 0x0 TcpLen: 20

=====
=====

02/11-15:21:44.865769 192.168.200.10:1538 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7260 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xACBA803E Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:21:44.865773 192.168.200.30:6667 -> 192.168.200.10:1538
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xACBA803F Win: 0x0 TcpLen: 20

=====
=====

02/11-15:21:45.331742 192.168.200.10:1538 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7261 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xACBA803E Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:21:45.331747 192.168.200.30:6667 -> 192.168.200.10:1538
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xACBA803F Win: 0x0 TcpLen: 20

=====
=====

02/11-15:22:18.310308 192.168.200.10:1539 -> 192.168.200.30:9999
TCP TTL:128 TOS:0x0 ID:7262 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xAD271E65 Ack: 0x0 Win: 0xFAF0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
=====

02/11-15:22:18.310312 192.168.200.30:9999 -> 192.168.200.10:1539
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xAD271E66 Win: 0x0 TcpLen: 20

=====
=====

02/11-15:22:18.796628 192.168.200.10:1539 -> 192.168.200.30:9999

31 36 33 39 20 2C 20 36 36 36 37 0D 0A 1639 , 6667..

=====
=====

02/14-10:48:03.351272 192.168.200.10:113 -> 192.168.200.30:1044
TCP TTL:128 TOS:0x0 ID:7922 IpLen:20 DgmLen:52 DF
A Seq: 0xF7861E8D Ack: 0x7B5D6C21 Win: 0xFAE3 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1072705 4212647

=====
=====

02/14-10:48:03.351319 192.168.200.10:1639 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7923 IpLen:20 DgmLen:40 DF
A Seq: 0xF78582C6 Ack: 0x7BB18C35 Win: 0xFAC2 TcpLen: 20

=====
=====

02/14-10:48:03.351331 192.168.200.30:6667 -> 192.168.200.10:1639
TCP TTL:64 TOS:0x0 ID:52129 IpLen:20 DgmLen:73 DF
AP Seq: 0x7BB18C35 Ack: 0xF78582C6 Win: 0x16D0 TcpLen: 20
4E 4F 54 49 43 45 20 41 55 54 48 20 3A 2A 2A 2A NOTICE AUTH :***
20 43 68 65 63 6B 69 6E 67 20 49 64 65 6E 74 0D Checking Ident.
0A

=====
=====

02/14-10:48:03.455133 192.168.200.10:1639 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7924 IpLen:20 DgmLen:116 DF
AP Seq: 0xF78582C6 Ack: 0x7BB18C56 Win: 0xFAA1 TcpLen: 20
55 53 45 52 20 4B 72 4A 68 56 6F 46 4E 46 78 4F USER KrJhVoFNFXO
6F 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A 4D o localhost 0 :M
4F 6E 45 41 49 67 47 70 52 67 54 51 4A 45 4C 63 OnEAlgGpRgTQJELc
54 47 4A 51 67 43 4C 41 4E 58 0A 4E 49 43 4B 20 TGJQgCLANX.NICK
6B 43 70 57 42 6F 57 4D 79 47 74 0A kCpWBoWMyGt.

=====
=====

02/14-10:48:03.455138 192.168.200.30:6667 -> 192.168.200.10:1639
TCP TTL:64 TOS:0x0 ID:52130 IpLen:20 DgmLen:40 DF
A Seq: 0x7BB18C56 Ack: 0xF7858312 Win: 0x16D0 TcpLen: 20

=====
=====

02/14-10:48:03.455140 192.168.200.10:113 -> 192.168.200.30:1044
TCP TTL:128 TOS:0x0 ID:7925 IpLen:20 DgmLen:86 DF
AP Seq: 0xF7861E8D Ack: 0x7B5D6C21 Win: 0xFAE3 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1072705 4212647
31 36 33 39 20 2C 20 36 36 36 37 20 3A 20 55 53 1639 , 6667 : US
45 52 49 44 20 3A 20 55 4E 49 58 20 3A 20 48 63 ERID : UNIX : Hc
4D 0A M.

=====
=====

02/14-10:48:03.455153 192.168.200.30:1044 -> 192.168.200.10:113
TCP TTL:64 TOS:0x0 ID:11191 IpLen:20 DgmLen:52 DF
A Seq: 0x7B5D6C21 Ack: 0xF7861EAF Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 4212658 1072705

=====
=====

02/14-10:48:03.455155 192.168.200.30:1044 -> 192.168.200.10:113
TCP TTL:64 TOS:0x0 ID:11192 IpLen:20 DgmLen:52 DF
AF Seq: 0x7B5D6C21 Ack: 0xF7861EAF Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 4212658 1072705

=====
=====

02/14-10:48:03.455156 192.168.200.30:6667 -> 192.168.200.10:1639
TCP TTL:64 TOS:0x0 ID:52131 IpLen:20 DgmLen:77 DF
AP Seq: 0x7BB18C56 Ack: 0xF7858312 Win: 0x16D0 TcpLen: 20
4E 4F 54 49 43 45 20 41 55 54 48 20 3A 2A 2A 2A NOTICE AUTH :***
20 47 6F 74 20 49 64 65 6E 74 20 72 65 73 70 6F Got Ident respo
6E 73 65 0D 0A nse..

=====
=====

02/14-10:48:03.455158 192.168.200.10:113 -> 192.168.200.30:1044
TCP TTL:128 TOS:0x0 ID:7926 IpLen:20 DgmLen:52 DF
A Seq: 0xF7861EAF Ack: 0x7B5D6C22 Win: 0xFAE3 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1072705 4212658

=====
=====

02/14-10:48:03.455160 192.168.200.10:113 -> 192.168.200.30:1044
TCP TTL:128 TOS:0x0 ID:7927 IpLen:20 DgmLen:52 DF
AF Seq: 0xF7861EAF Ack: 0x7B5D6C22 Win: 0xFAE3 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1072705 4212658

=====
=====

02/14-10:48:03.455171 192.168.200.30:1044 -> 192.168.200.10:113
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
A Seq: 0x7B5D6C22 Ack: 0xF7861EB0 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 4212658 1072705

=====
=====

02/14-10:48:03.608206 192.168.200.10:1639 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7928 IpLen:20 DgmLen:40 DF
A Seq: 0xF7858312 Ack: 0x7BB18C7B Win: 0xFA7C TcpLen: 20

=====
=====

02/14-10:48:32.217142 192.168.200.30:6667 -> 192.168.200.10:1639
TCP TTL:64 TOS:0x0 ID:52132 IpLen:20 DgmLen:89 DF
AP Seq: 0x7BB18C7B Ack: 0xF7858312 Win: 0x16D0 TcpLen: 20
4E 4F 54 49 43 45 20 41 55 54 48 20 3A 2A 2A 2A NOTICE AUTH :***
20 43 6F 75 6C 64 6E 27 74 20 6C 6F 6F 6B 20 75 Couldn't look u
70 20 79 6F 75 72 20 68 6F 73 74 6E 61 6D 65 0D p your hostname.
0A

=====
=====

02/14-10:48:32.377308 192.168.200.10:1639 -> 192.168.200.30:6667
TCP TTL:128 TOS:0x0 ID:7930 IpLen:20 DgmLen:40 DF
A Seq: 0xF7858312 Ack: 0x7BB18CAC Win: 0xFA4B TcpLen: 20

=====
=====

02/14-10:48:32.856066 192.168.200.30:6667 -> 192.168.200.10:1639
TCP TTL:64 TOS:0x0 ID:52133 IpLen:20 DgmLen:1064 DF
AP Seq: 0x7BB18CAC Ack: 0xF7858312 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 6B 43 70 57 42 domain 001 kCpWB
6F 57 4D 79 20 3A 57 65 6C 63 6F 6D 65 20 74 6F oWMy :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 6B 43 70 57 lay Network kCpW
42 6F 57 4D 79 0D 0A 3A 6C 6F 63 61 6C 68 6F 73 BoWMy...localhos
74 2E 6C 6F 63 61 6C 64 6F 6D 61 69 6E 20 30 30 t.localdomain 00
32 20 6B 43 70 57 42 6F 57 4D 79 20 3A 59 6F 75 2 kCpWBoWMy :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 6B 43 70 57 42 6F 57 4D 79 20 OTICE kCpWBoWMy
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
73 69 6F 6E 20 32 2E 38 2F 68 79 62 72 69 64 2D sion 2.8/hybrid-
36 2E 33 2E 31 0D 0A 3A 6C 6F 63 61 6C 68 6F 73 6.3.1...localhos
74 2E 6C 6F 63 61 6C 64 6F 6D 61 69 6E 20 30 30 t.localdomain 00
33 20 6B 43 70 57 42 6F 57 4D 79 20 3A 54 68 69 3 kCpWBoWMy :Thi
73 20 73 65 72 76 65 72 20 77 61 73 20 63 72 65 s server was cre
61 74 65 64 20 54 75 65 20 4A 75 6E 20 34 20 32 ated Tue Jun 4 2
30 30 32 20 61 74 20 31 36 3A 35 39 3A 34 35 20 002 at 16:59:45
45 44 54 0D 0A 3A 6C 6F 63 61 6C 68 6F 73 74 2E EDT...localhost.
6C 6F 63 61 6C 64 6F 6D 61 69 6E 20 30 30 34 20 localdomain 004
6B 43 70 57 42 6F 57 4D 79 20 6C 6F 63 61 6C 68 kCpWBoWMy localh
6F 73 74 2E 6C 6F 63 61 6C 64 6F 6D 61 69 6E 20 ost.localdomain
32 2E 38 2F 68 79 62 72 69 64 2D 36 2E 33 2E 31 2.8/hybrid-6.3.1
20 6F 4F 69 77 73 7A 63 72 6B 66 79 64 6E 78 62 oOiwszcrkfydnxb
20 62 69 6B 6C 6D 6E 6F 70 73 74 76 65 0D 0A 3A biklmnopstve...:
6C 6F 63 61 6C 68 6F 73 74 2E 6C 6F 63 61 6C 64 localhost.locald
6F 6D 61 69 6E 20 30 30 35 20 6B 43 70 57 42 6F omain 005 kCpWBo
57 4D 79 20 57 41 4C 4C 43 48 4F 50 53 20 50 52 WMy WALLCHOPS PR
45 46 49 58 3D 28 6F 76 29 40 2B 20 43 48 41 4E EFIX=(ov)@+ CHAN
54 59 50 45 53 3D 23 26 20 4D 41 58 43 48 41 4E TYPES=#& MAXCHAN
4E 45 4C 53 3D 32 30 20 4D 41 58 42 41 4E 53 3D NELS=20 MAXBANS=
32 35 20 4E 49 43 4B 4C 45 4E 3D 39 20 54 4F 50 25 NICKLEN=9 TOP
49 43 4C 45 4E 3D 31 32 30 20 4B 49 43 4B 4C 45 ICLEN=120 KICKLE
4E 3D 39 30 20 4E 45 54 57 4F 52 4B 3D 45 46 6E N=90 NETWORK=EFn
65 74 20 43 48 41 4E 4D 4F 44 45 53 3D 62 2C 6B et CHANMODES=b,k
2C 6C 2C 69 6D 6E 70 73 74 20 4D 4F 44 45 53 3D ,l,imnpst MODES=
34 20 3A 61 72 65 20 73 75 70 70 6F 72 74 65 64 4 :are supported
20 62 79 20 74 68 69 73 20 73 65 72 76 65 72 0D by this server.
0A 3A 6C 6F 63 61 6C 68 6F 73 74 2E 6C 6F 63 61 .:localhost.loca
6C 64 6F 6D 61 69 6E 20 32 35 31 20 6B 43 70 57 ldomain 251 kCpW
42 6F 57 4D 79 20 3A 54 68 65 72 65 20 61 72 65 BoWMy :There are
20 30 20 75 73 65 72 73 20 61 6E 64 20 32 20 69 0 users and 2 i
6E 76 69 73 69 62 6C 65 20 6F 6E 20 31 20 73 65 nvisible on 1 se

====+

© SANS Institute 2005, Author retains full rights.

Appendix H BinText output for unpacked malware

```
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 already 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
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  <$hE:@
00003ABB 00403ABB 0  PhR:@
00003B99 00403B99 0  nick.pre
00003BA2 00403BA2 0  %s-%04u
00003BAA 00403BAA 0  irc.user
00003BB3 00403BB3 0  irc.usereal
00003BBF 00403BBF 0  irc.real
00003BC8 00403BC8 0  irc.pass
00003BE0 00403BE0 0  tsend(): connection to %s:%u failed
00003C20 00403C20 0  USER %s localhost 0 :%s
00003C38 00403C38 0  NICK %s
00003DF5 00403DF5 0  Ph <@
000040BF 004040BF 0  PRIVMSG
00004100 00404100 0  trevc(): Disconnected from %s err:%u
0000446B 0040446B 0  NOTICE
00004472 00404472 0  %s %s :%s
00004615 00404615 0  Ph}D@
00004711 00404711 0  MODE %s -o+b %s *@%s
00004798 00404798 0  C'PSWh
000047B4 004047B4 0  Sh'G@
000047E7 004047E7 0  MODE %s -bo %s %s
0000487B 0040487B 0  Sh'G@
00004924 00404924 0  %s.key
00004A63 00404A63 0  Ph'G@
00004AA8 00404AA8 0  sk#%u %s is dead!
00004ABA 00404ABA 0  s_check: %s dead? pinging...
00004AD7 00404AD7 0  PING :ok
00004B00 00404B00 0  s_check: send error to %s disconnecting
00004B28 00404B28 0  expect the worst
00004B39 00404B39 0  s_check: killing socket %s
00004B54 00404B54 0  irc.knick
00004B5E 00404B5E 0  jtr.%u%s.iso
00004B6B 00404B6B 0  ison %s
00004B74 00404B74 0  servers
00004B7C 00404B7C 0  s_check: trying %s
00004DAA 00404DAA 0  Ph9K@
00004ED5 00404ED5 0  PhkK@
00004F41 00404F41 0  ShtK@
00004FD8 00404FD8 0  uYVh|K@
00005052 00405052 0  %s.mode
0000505A 0040505A 0  MODE %s %s
00005078 00405078 0  ShRP@
000050DA 004050DA 0  Sh$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 004057B3 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
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>

```

```

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 name 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 0040806B 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 sock
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 00408B90 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
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 004093CC 0 ?free
000093D7 004093D7 0 ?update
000093DF 004093DF 0 ?hostname
000093EE 004093EE 0 ?!fff
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=
00009A16 00409A16 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 0040A65C 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$W8kl8Jr1X8DOHZsmlp9qq0
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
0000C0BA 0040C0BA 0 SSL_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 QueryPerformanceCounter
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 0040C897 0 RII enhanced drive
0000C8C0 0040C8C0 0 software\microsoft\windows\currentversion\run
0000C8EE 0040C8EE 0 /d "%s"
0000CE3D 0040CE3D 0 < u&
0000D010 0040D010 0
./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
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
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
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

```

```

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
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
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 built-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
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
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 CreateFileA
0011B82A 0051B82A 0 CreateMutexA
0011B83A 0051B83A 0 CreatePipe
0011B84A 0051B84A 0 CreateProcessA
0011B85E 0051B85E 0 CreateToolhelp32Snapshot
0011B87A 0051B87A 0 DeleteFileA
0011B88A 0051B88A 0 DuplicateHandle
0011B89E 0051B89E 0 EnterCriticalSection
0011B8B6 0051B8B6 0 ExitProcess
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 GetSystemDirectoryA
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
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 _setmode
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
0011BDDBA 0051BDDBA 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
0011BEC6	0051BEC6	0	PeekMessageA
0011BED6	0051BED6	0	GetFileVersionInfoA
0011BEEE	0051BEEE	0	VerQueryValueA
0011BF02	0051BF02	0	InternetCloseHandle
0011BF1A	0051BF1A	0	InternetGetConnectedState
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	setsockopt
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
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 D4I|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 msvcr.dll
0011E08C 0051E08C 0 msvcr.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.

Appendix I Commands Test

```
?status
service:N user:Rick inet connection:Y contype: Lan reboot privs:Y
?!fif
?dcc
?free
usage: ?free <cmd>
?killsk
unable to close socket 4017464
?part
_Set an irc sock to preform ?part command on_
_Type _sklist_ to view current sockets, then _dccsk_ <#>
?set
set jtr.bin msrll.exe
set jtr.home mfm
set bot.port 2200
set jtr.id run5
set irc.quit
set servers collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:8080
set irc.chan #mils
set pass $1$KZLPLKDF$W8kl8Jr1X8DOHZsmIp9qq0
set dcc.pass $1$KZLPLKDF$55isA1ITvamR7bjAdBziX.
?uptime
sys: 5d 08h 24m 12s bot: 01h 08m 52s
?akick
?dccsk
usage ?dccsk <socks #>
?get
_Set an irc sock to preform ?get command on_
_Type _sklist_ to view current sockets, then _dccsk_ <#>
?login
?play
(null): somefile
?si
WINXP (u:Rick) mem:(230/383) 39% GenuineIntel Intel(R) Pentium(R) M processor 1500MHz
?wget
?aop
?del
?hostname
host: testlab-XP ip: 192.168.200.10
?ls
02/23/2005 02:01 <DIR> .
02/23/2005 02:01 <DIR> ..
02/23/2005 03:12      1084 jtram.conf
01/14/2005 20:39      1175552 msrll.exe
?ps
0  [System Process]
4  System
472 smss.exe
620 csrss.exe
644 winlogon.exe
688 services.exe
700 lsass.exe
864 svchost.exe
956 svchost.exe
```

```

1180 svchost.exe
1216 svchost.exe
1344 explorer.exe
1472 spoolsv.exe
1664 VMwareTray.exe
1680 VMwareUser.exe
1688 msmsgs.exe
1912 wdfmgr.exe
1980 VMwareService.exe
1972 wuauclt.exe
928 bintext.exe
1112 cmd.exe
1804 procepx.exe
1176 ldag.exe
744 OLLYDBG.EXE
1164 msrll.exe
?sklist
#1 [fd:352] 192.168.200.50:0 [DCC ICON RNL ] last:0
|=> (notpasswdnotpasswd) (00000021)
?cd
?die
?hush
_Set an irc sock to preform ?hush command on_
_Type _sklist_ to view current sockets, then _dccsk_ <#>
?md5p
?md5p <pass> <salt>
?pwd
C:\WINDOWS\system32\mfmm
?ssl
?ssl: -1
?clone
usage ?clone: server[:port] amount
?dir
02/23/2005 02:01 <DIR> .
02/23/2005 02:01 <DIR> ..
02/23/2005 03:23 1084 jtram.conf
01/14/2005 20:39 1175552 msrll.exe
05/10/2004 22:29 41984 msrll.orig.exe
01/14/2005 21:03 655360 unpacked.id0
01/14/2005 21:03 4669440 unpacked.id1
01/14/2005 21:03 8192 unpacked.nam
?join
_Set an irc sock to preform ?join command on_
_Type _sklist_ to view current sockets, then _dccsk_ <#>
?mkdir
?raw
_Set an irc sock to preform ?raw command on_
_Type _sklist_ to view current sockets, then _dccsk_ <#>
?clones
?clones: [NETWORK|all] <die|join|part|raw|msg> <"parm"> ...
?dump
?jump
?move
?status
service:N user:Rick inet connection:Y contype: Lan reboot privs:Y
?sums
jtram.conf 0362caeee75fc97e56da5628fd3c1d42
msrll.exe d05c747e2158eb2b50643fee5c4ad338

```

?con
?echo
(null)
?kb
Set an irc sock to preform ?kb command on
_Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?msg
Set an irc sock to preform ?msg command on
_Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?rmdir
?uattr
Set an irc sock to preform ?uattr command on
_Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?copy
?exec
?kill
?nick
Set an irc sock to preform ?nick command on
_Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?run
?unset
?fif
?killall
?op
Set an irc sock to preform ?op command on
_Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?say hi
Set an irc sock to preform ?say command on
_Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?update
?update: <url> <id>
?reboot
later!

© SANS Institute 2005, Author retains full rights.