

The questions...

**\* Describe your malware lab.**

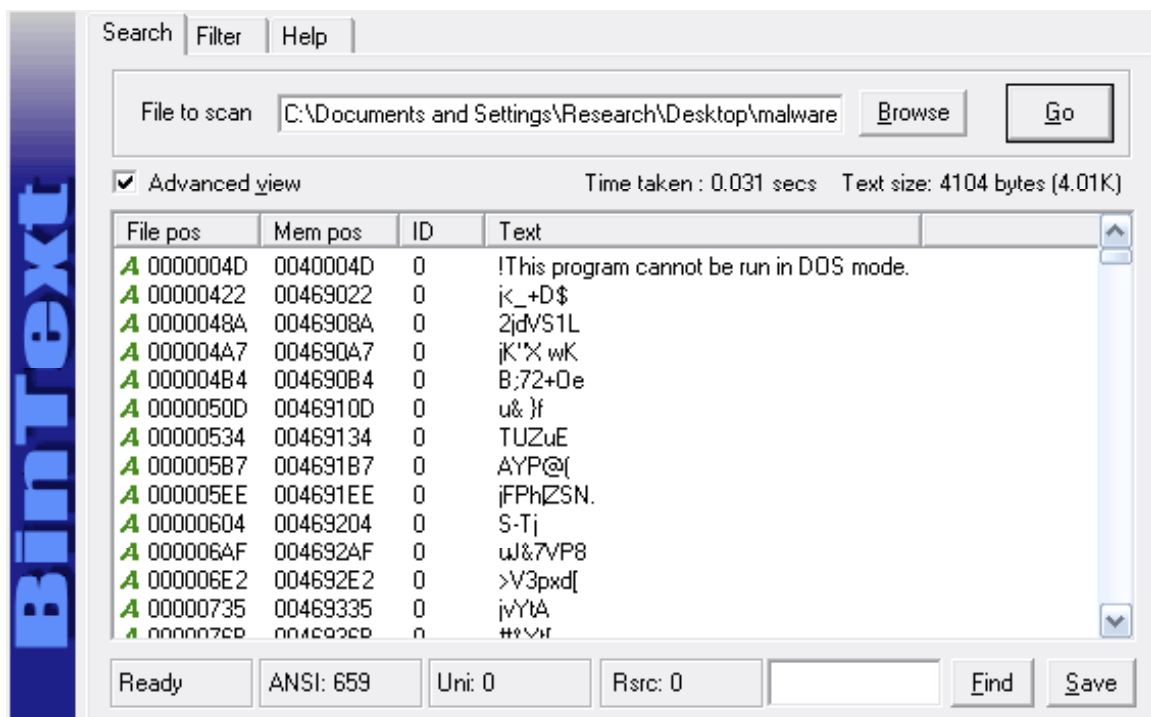
For my lab I'm using VMware Fusion with a host operating system of Mac OS X. My VMware's operating system is Windows XP SP 1. My host computer is connected via a wireless router. There are no other computers connected to the network.

**\* What information can you gather about the malware without executing it?**

Using the standard Microsoft Windows Properties option of right clicking on a file is of no real use to anyone analyzing a file. When I check the Properties tab I can see the Created, Modified, and Accessed dates are all the date that I un-zipped our malware.exe. In this case the right-click view Properties is of little use for any prevalent information about the executable. In order to find more information about the executable it is extremely helpful to use some tools to dig deeper.

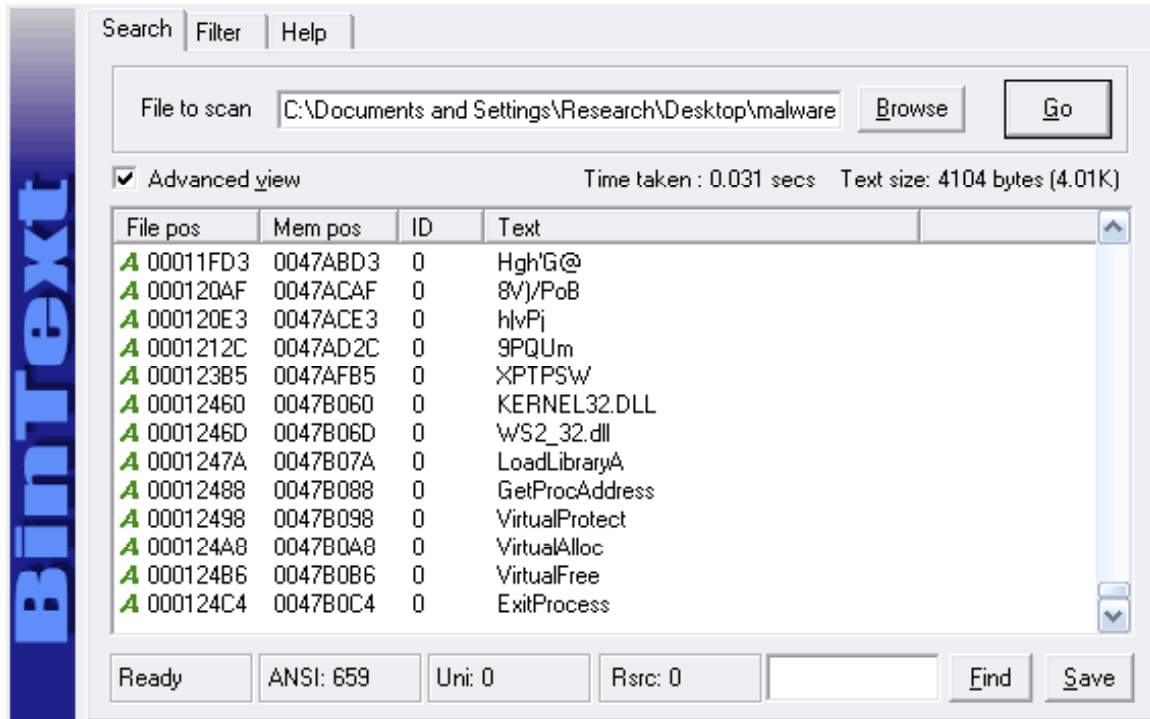
The three tools that I started with are Bintext by Foundstone, PeView by Wayne J. Radburn and Google. These are good tools for trying to find some quick information about the file that could quickly narrow down the broad spectrum of possibilities that are contained inside of a compiled executable.

The first tool I used was BinText by FoundStone. Bintext is a small text extractor that can recover any Ascii, Unicode and other usefull strings. A strings extractor is helpful in getting some basic clues about the executable. By glancing at the strings we can see "This program cannot be run in DOS mode". This is part of the MS-DOS Stub program that makes up a component of the PE-File format. The PE-File format can be thought upon as the skeleton of a compiled Microsoft Windows file.



The above image is our malware.exe strings as viewed through BinText.

Below the MS-DOS stub we can see random strings that don't have any particular order or sequence. This pattern of entropy or randomization is usually typical in a compressed file. If I keep scrolling down the file I can see the pattern of random sequences continues almost in an orderly fashion all the way to the bottom of the executable.



The above image is the bottom strings of the executable.

After scrolling to the bottom of the executable's strings I can see the name of two DLLs and some APIs. The two DLLs are standard Microsoft Windows system files. Kernel32.dll handles memory management, input/output and interrupts. WS2\_32.dll contains the Windows Socket API. A socket is used for network communication and is commonly used for setting up a client-server application. The APIs are LoadLibrary, GetProcAddress, VirtualProtect, VirtualAlloc, VirtualFree and ExitProcess.

After Googling the APIs I can tell these APIs are mainly for loading libraries, locating API addresses and memory management but nothing functionally useful. There is no input/output, networking communication, open/close, graphics functions or anything that would motivate someone to program this executable. The starkness of API calls makes me wonder if there is something missing. After viewing the strings of the executable I opened the file in PeView.

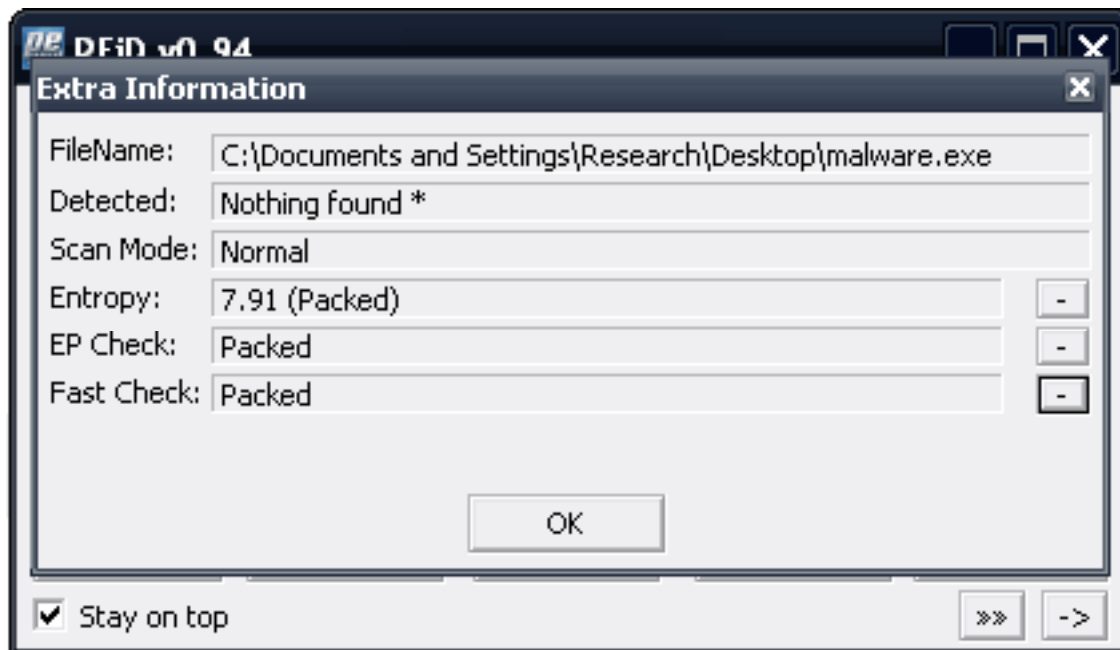
PeView is a tool used to view the structure and content of the PE-File Format. Using PeView I can view the entry point of the executable, the compilation date through

the Time Date Stamp, section names, and other information that can give me clues if the file is packed or not. Some flags that stand out for me is the “Base of Code” in the PE. Typical base is 00001000 but in our file the base is 0006900. The Section names ABC0, ABC1, ABC2 look arbitrary and not of any standard compiled format. When reviewing Section ABC0 I found that the section is empty.

**\* Is the malware packed? If so, how did you determine what it was?**

The lack of recognizable strings, useful APIs, and standard PE settings make me suspect the file is packed. To help confirm my guess I opened the file in PEID.

PEID works similar to an Anti-Virus engine. PEID has a database of known packer signatures; it then scans the file searching for known signatures and returns the packer if a signature is found. Signature database are reliable for known packers but custom and one-off packers can easily evade signature scanners. In my instance no signature was found. A useful feature of PEID is its “Extra Information” option.



The above image is of PEID’s Extra Information Window “>> >>”

The “Extra Information” option uses a heuristics approach to estimate if the file is packed or not. PEID also confirms that the file is packed.

```
0047AFC5 | . | 39C4 | CMP ESP,EAX
0047AFC7 | .^ | 75 FA | JNZ SHORT malware.0047AFC3
0047AFC9 | . | 83EC 80 | SUB ESP,-80
0047AFCC | .- | E9 FB59F9FF | JMP malware.004109CC
0047AFD1 | | 00 | DB 00
0047AFD2 | | 00 | DB 00
```

If I was to guess I would say the file is packed with a modified version of UPX. The above code is usually used to jump to the OEP (Original Entry Point) of a packed UPX file.

**\* Describe the malware's behavior. What files does it drop? What registry keys does it create and/or modify? What network connections does it create? How does it auto-start, etc?**

When the malware is first executed it checks to see if it's running from the "C:\Windows". If the file is not running from the Windows directory it copies itself to the Windows directory and renames itself "Winsec32.exe". The malware.exe will execute the copied file and then exit process.

The executable writes the following Registry keys to insure it will be executed on reboot.

```
HKEY_CURRENT_USER\Software\Microsoft\OLE    Microsoft Svchost local services
        "Winsec32.exe"
```

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunSer
vices    Microsoft Svchost local services    "Winsec32.exe"
```

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
        Microsoft Svchost local services    "Winsec32.exe"
```

```
HKEY_USERS\S-1-5-21-1275210071-343818398-682003330-
1004\Software\Microsoft\OLE    Microsoft Svchost local services
        "Winsec32.exe"
```

For tracking Registry writes I used InstallWatch by Epsilon Squared. I took a snapshot of the Registry, ran malware.exe in Kerberos API spy and then analyzed the snapshot. I used Kerberos API so I had a log of all API and their arguments. I used the log of InstallWatch and Kerberos API spy log to cross-reference and verify that the executable did write the Registry keys.

The executable then tried to connect to an IRC channel at "testirc1.sh1xy2bg.NET" with a channel name of "#challenge" using "happy12" as a password with the following parameters "NICK USA[XP]5670302 USER usonksbd 0 0 :USA[XP]5670302I4B". The URL would not resolve and the executable continued (loop then sleep) trying to connect to the un-resolvable URL. Since the URL would not resolve I had to step through the executable using Olydbg rather than sniff the network using Wireshark to get the parameters.

**\* What type of command and control server does the malware use? Describe the server and interface this malware uses as well as the domains and URLs accessed by the malware.**

The control server is an IRC server at "testirc1.sh1xy2bg.NET" with a channel name of "#challenge" and password of "happy12".

**\* What commands are present within the malware and what do they do? If possible, take control of the malware and run some of these commands, documenting how you did it.**

Unfortunately I was unable to take control of the malware. I'm really looking forward to reading on how others might have accomplished this. I thought of setting up an IRC server and changing arguments on the stack but due to time constraints I was unable to accomplish this.

**\* How would you classify this malware? Why?**

I would classify the executable as a backdoor IRC Bot. The executable looks to be multifunctional it can connect to an IRC server, download/upload files, keylog, open a remote shell, kill processes, "flood" IPs, scan and execute a VNC exploit and many other generic administrative bot/backdoor tasks.

**\* What do you think the purpose of this malware is?**

It's hard to say specifically what the purpose of the malware is but it looks to run as a generic bot. I was only able to walk through the code with a debugger patching some of the flow of the code. After reviewing the configuration file for the bot code (Sorry I cheated and jumped a head to a bonus question) the malware looks to be a bot with all the bells and whistles compiled in by default. By glancing at the Headers folder I can see ddos.h, download.h, icmpflood.h, passwd.h, keylogger.h and many others. Considering the lack of configurations makes me wonder if this bought is for the newbies.

**Bonus questions: (These questions are not required to be answered but could be used to break a tie for prizes.)**

**\* Is it possible to find the malware's source code? If so, how did you do it?**

Yes, it is possible to find the malware's source code.

A	00023428	00423428	0	RealmBoT (irc.p.l.g) .
A	00023440	00423440	0	. Bot started.
A	00023450	00423450	0	%s %d "%s"
A	0002345C	0042345C	0	%s\%s
A	0002346C	0042346C	0	RealmBoT (irc.p.l.g) .
A	00023484	00423484	0	. Connected to %s.
A	00023498	00423498	0	NICK %s
A	000234A1	004234A1	0	USER %s 0 0 :%s
A	000234B4	004234B4	0	PASS %s
A	000234C0	004234C0	0	MODE %s %s
A	000234D0	004234D0	0	USERHOST %s
A	000234E0	004234E0	0	RealmBoT (irc.p.l.g) .
A	000234F8	004234F8	0	. User: %s logged in.
A	00023510	00423510	0	[REALMBOT] : Thank for trying.
A	00023530	00423530	0	RealmBoT (irc.p.l.g) .
A	00023548	00423548	0	. *Failed host auth by: (%s!%s).
A	0002356C	0042356C	0	NOTICE %s :Orders: No Talk with you.
A	00023594	00423594	0	NOTICE %s :WTF!? no yet f :er!. (%s!%s).
A	000235C0	004235C0	0	RealmBoT (irc.p.l.g) .
A	000235D8	004235D8	0	. *Failed pass auth by: (%s!%s).
A	000235FC	004235FC	0	NOTICE %s :No pass for you.
A	0002361C	0042361C	0	NOTICE %s :Are you a Ft er?. (%s!%s).

While searching through the strings of the dumped executable I found the string "RealmBoT" with the strings below "Bot started". I decided to Google the strings "RealmBot Source Code" and the source code was the fifth search result. Thankfully the bot's author was very descriptive and didn't allow any ambiguity with this particular bot's name.

**\* How would you write a custom detection and removal tool to determine if the malware is present on the system and remove it?**

If we are only worried about this particular variant we could do an MD5 hash of the executable and then use a tool like ClamWin to scan the computer searching for the file hash. ClamAv has a pdf available called "Creating signatures for ClamAV" that is extremely easy to use as a reference for creating virus signatures. I have never created a signature for ClamWin before but it took only a couple of seconds to mimic the text in the pdf to create a signature.

```

C:\WINDOWS\System32\cmd.exe
C:\Program Files\ClamWin\bin>sigtool --md5 "C:\Documents and Settings\Research\Desktop\Malware_a\malware.exe" > test.hdb
C:\Program Files\ClamWin\bin>clamscan -r -i -d test.hdb "C:"_

```

The first line creates the MD5 hash of the "malware.exe" and saves it in "test.hdb"

```
"C:\Program Files\ClamWin\bin>sigtool --md5 "C:\Documents and Settings\Research\Desktop\Malware_a\malware.exe" > test.hdb"
```

The second line recursively (-r) scans the computer searching for the md5 hash (-d test.hdb) of malware.exe and only displaying the infected (-i) executable.

```
"C:\Program Files\ClamWin\bin>clamscan -r -i -d test.hdb "C:"
```

This will only display files that have the same MD5 hash of malware.exe. It will not remove it.

Luckily the malware's MD5 hash remains static and does not change. If by chance the malware.exe did modify its code or the packer changed some bytes or by chance one byte was changed the MD5 hash would no longer match. If this were the case with malware.exe I would have to create a hash that was more specifically based off a section of code rather than the hash of the whole executable. Sectional MD5s hash scanning is an option in ClamWin but due to the file being packed it is likely that we could be hashing a section of code that is not malicious such as UPX packer code.

One issue with scanning malware.exe for a hash is that the file is packed. In order to find a hash that is not the packer's code or the original code compressed we will have to unpack the code without executing malicious code. This is where anti-virus companies start using emulators, custom un-packer scripts and sandbox tools. This starts to get extremely complicating but fascinating. Luckily (I think) ClamWin has some custom scripts that can unpack FSG, UPX and a couple of other common packers.

In order to create a signature off a specific piece of code an individual has to spend a substantial amount of time reverse engineering the code in order to create a signature off of code that is not a common library, not generic code, not likely to change by being recompiled with a different setting and won't give a false positive. I choose this piece of code for my signature. I used the freeware version of IDA to for searching for a good spot for a signature.

```

sub     eax, 4
pop     ecx
jz      short loc_403991

; CODE XREF: I
call    _rand
push   1Ah
cdq
pop     ecx
idiv   ecx
push   edi ; Str
add    dl, 97
mov    Str[ebx], dl
inc    ebx
call   _strlen
sub    eax, 4
pop    ecx
cmp    ebx, eax
jb     short loc_40396E

; CODE XREF: I
; Disney_Land-
lea    eax, [ebp+CurrentDirectory]
push   edi
push   eax

```

My signature of the code is:

83E804597423E8????????6A1A9959F7F95780C26188932CC8410043E8????????83  
E804593BD872DD8D8518FCFFFF57 with an offset at 00403968. I d

The question marks are wild cards for the API addresses.

Thanks to everyone over at MalwareChallenge.com! This was the first piece of malware that I have analyzed in over a year and I really enjoyed it. Can't wait for the next challenge.

Alexander Hanel

P.S.

Sorry for the brief ending the deadline is in thirty minutes for this paper ☺