PRACTICAL MALWARE ANALYSIS

Kris Kendall
kris.kendall@mandiant.com

WHY PERFORM MALWARE ANALYSIS?

What are some of the reasons that one might want to invest the (sometimes significant) resources required to effectively analyze malware? Imagine that you are in the unenviable position of finding some unknown, running, and potentially malicious executable program on an important server. In this situation, some very important questions can be answered—and usually, can *only* be answered—by conducting malware analysis.

Malware analysis can be conducted with a variety of goals in mind. Some of the common reasons that you might want to analyze a malicious program include:

- To assess damage from an intrusion
- To discover and catalogue indicators of compromise that will reveal other machines that have been affected by the same malware or intruders
- To determine the sophistication level of the malware author
- To identify the vulnerability that was exploited to allow the malware to get there in the first place
- To identify the intruder or insider that is responsible for installing the malware
- To learn and have fun!

By extending a common definition of the word "analysis", we define malware analysis as "the action of taking malware apart to study it". While you are studying the malware, your purpose is to discover the answers to questions about the malware. These questions can be broken down into "business" questions and "technical" questions. Some of the most common business questions answered by malware analysis are:

1. What is the purpose of the malware?
2. How did it get here?
3. Who is targeting us and how good are they?
4. How can I get rid of it?
5. What did they steal?
6. How long has it been here?
7. Does it spread on its own?
8. How can I find it on other machines?
9. How do I prevent this from happening in the future?

Answers to these business questions are usually revealed by combining and synthesizing details revealed by asking purely technical questions, such as:

1. What are the network-based indicators that reveal the presence and activity of the malware?

2. What are the host-based indicators that reveal the presence and activity of the malware?
3. Is the malware persistent? If so, what mechanism does it use to ensure that it keeps running after a machine is rebooted?
4. When was the program written, compiled, and installed?
5. Is the program based on any other well-known tool?
6. What language was used to write the program?
7. Is the program packed? What program was used to pack it?
8. Does the program have any anti-debugging functionality?
9. Does the program include any rootkit functionality?

CREATING A SAFE AND EFFECTIVE ANALYSIS ENVIRONMENT

Malware is software that is explicitly designed to *perform evil*. This means that it is generally a bad idea to let malware run on the same PC on which you send e-mail to your friends, do you online banking, and write papers for security conferences. One solution to this problem is to create an "analysis lab" consisting of a bunch of computers that are on their own physically partitioned network. These machines should have a standardized software build that can easily be restored from a backup image after some piece of malware has finished destroying the system. However, it is much easier (and only somewhat less safe) to use virtual machines to create a simulated lab environment. There are several software products (some of them free) that can be used to create virtual machines. VMware is currently my favorite for malware analysis by virtue of its ability to create a tree of snapshots that capture system state at various times. These snapshots can be used to easily revert to a previous system state (such as right before you double-clicked on the icon for `rustock.exe`). See Figure 1 for an example of how easy it is to keep a nested tree of system states that allows you to virtually move forward and back in the history of your virtual machine's state.

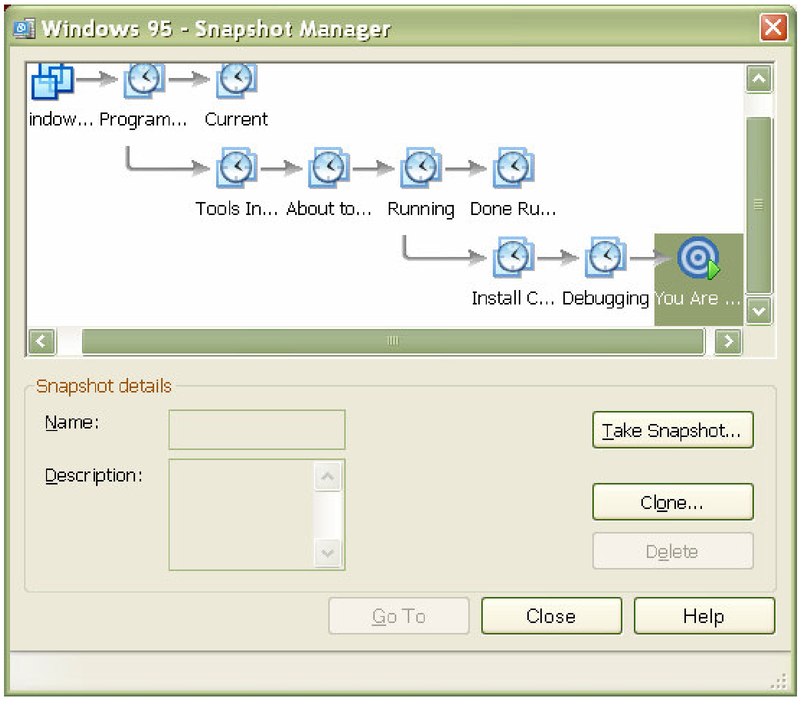


Figure 1: VMware's tree of snapshots

Although VMware is (in my opinion) the best virtualization platform for malware analysis, there are several other good options, including Parallels, Microsoft Virtual PC, and Xen.

Though using a virtualized victim machine provides some level of control over the behavior of the malware, there are a few "gotchas" associated with running malware in a virtual machine:

1. Your virtualization software is not perfect, and may allow information to "leak" from the virtual machine to your host machine in ways that you didn't expect.
2. Malicious code can detect that it is running in a virtual machine and may modify its behavior.
3. A 0-day worm that can exploit a listening service on your host OS will escape the virtual machine sandbox, even if you are using host-only networking!

If you are setting up a dedicated malware analysis environment, there are several tools that can make the job of re-imaging machines easier:

- If you have a budget, Norton Ghost works just fine for quickly restoring system images.
- If you don't have a budget, but like the features of Ghost, check out `udpcast`[1]. It works great, and the price (free) is hard to beat.
- Joe Stewart (from Lurhq) has developed an automated system called `Truman`[2] that is specifically designed for malware analysis using a pair of physical machines.
- CoreProtect makes a piece of hardware called a hard drive write cache card that can be used to set up a system that restores itself to a pristine state each time it is rebooted (Figure 2).

.
Figure 2: CoreRestore card from CoreProtect

Once you have chosen whether to use virtual machines or physical machines for your malware analysis, the next choice is what level of network access you want to allow the machines to have. It is easier and faster to conduct malware analysis using a victim machine that is connected to the Internet and is able to connect to the *real* controlling hosts being operated by the intruder. However, this approach has several significant drawbacks:

1. The attacker might change his behavior when he sees connections from a machine that he didn't hack.
2. By allowing malware to connect to a controlling server, you may be entering a real-time battle with an actual human for control of your analysis (virtual) machine.

[1] http://udpcast.linux.lu/
[2] http://www.lurhq.com/truman/

3. The external IP address used by your analysis machine may become the target for additional attacks.
4. If the malware spreads automatically or conducts DDoS attacks, you may end up unwittingly attacking others.

For these reasons, I usually recommend conducting malware analysis using a closed network with virtualized services (like DNS servers, HTTP server, etc.). This approach can require significant extra effort; in order to conduct effective dynamic analysis in a closed network environment you may need to reverse-engineer and recreate the functionality of the controlling server. This is not easy, but is usually worth it given the significant risks of allowing malware to access the Internet. If you do conduct malware analysis on a machine connected to the Internet, I suggest using a separate dedicated firewall with a very restrictive ruleset to ensure that you are aware of exactly what traffic you are allowing from the malware.

STATIC ANALYSIS TECHNIQUES

As I begin analysis of a suspected piece of malware, I usually start out by performing some initial static analysis. In essence, I "kick the tires" of the program I am examining—taking a look at some of its more obvious external features. This section focuses on static analysis techniques that do not require an extensive programming or reverse engineering background. Detailed static analysis of a program's internal logic will generally involve use of a disassembler and analysis of assembly language code—coverage of these more advanced techniques is outside the scope of this paper.

Static analysis is generally safer than dynamic analysis; because the code isn't actually running, you don't need to worry about it deleting files, calling home, or stealing data. Generally, the only risk involved in static analysis is the risk of accidentally double-clicking or otherwise accidentally running the malware. The risk of accidentally running malware can be reduced by conducting static analysis on a machine running a different operating system than the malware was designed to run; for example, static analysis of Windows malware can safely be conducted on an OS X system.

File Fingerprinting

Before doing anything else, it is advisable to compute a cryptographic hash value for each file under investigation. Although there are a wide variety of hash functions available, the best for the purpose of malware analysis is the one most likely to be used by other researchers—generally MD5, SHA1, or SHA256. After the file hash has been computed, you can also use the file hash to periodically verify that the program has been modified, or has modified itself. Many programs are available that can compute hash values for files. One of the most flexible is the open source command-line program `md5deep`[3] by Jesse Kornblum.

Virus Scanning

If the file being examined is a component of a well-known piece of malware, there is a chance that it will be recognized as such by anti-virus software. If the anti-virus program recognizes the malware, the anti-virus vendor will typically post analysis describing it. This analysis will sometimes provide only minimal details, but other times it will be quite thorough, including lengthy discussion of the software's capabilities, signatures, and instructions for removal. Clearly this information gives you a giant leg up in your analysis.

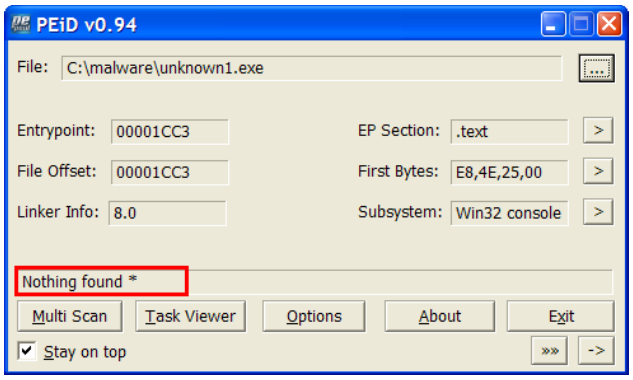
[3] http://md5deep.sourceforge.net

Websites like http://www.virustotal.com and http://virusscan.jotti.org allow you to upload files and have them scanned by a wide-variety of different scan engines. This is very convenient since most anti-virus programs will not allow installation on a machine alongside another anti-virus program. Keep in mind that these websites keep the files you upload and may share them with other people; the file you upload might have been customized for your environment or contain information about your organization that you wouldn't want to share with others.

Packer Detection

One of the major complicating factors in performing malware analysis is the proliferation of programs that modify an executable file to obfuscate its contents and hide the actual program logic from a reverse engineer performing static analysis. Programs that modify other program files to compress or disguise their contents are most commonly referred to as "executable packers" or just "packers". When a packer compresses, encrypts, or otherwise modifies an executable program, the program looks much different from the static analysis perspective, but still runs as it did before it was "packed". Once a program has been packed, the original program's logic and other metadata are very hard to recover through static analysis.

PEiD[4] is a free program that has signatures for over 600 different compilers and packers. To use PEiD, simply open a file with PEiD and take note of the value the PEiD reports in the text box highlighted in Figure 3. If PEiD reports that a file is "Not a valid PE file", then it didn't match the basic signature of a PE file. PEiD does a good job of identifying many compilers and packers. Sometimes PEiD will report "Nothing Found *". In this case, the file is a valid executable, but PEiD did not find the signature of a known compiler or packer.


PEiD v0.94

File: C:\malware\unknown1.exe

Entrypoint: 00001CC3
EP Section: .text
File Offset: 00001CC3
First Bytes: E8,4E,25,00
Linker Info: 8.0
Subsystem: Win32 console

Nothing found *

Figure 3: PEiD with an unknown compiler/packer

Mandiant has also developed and released a free tool[5] that uses a variety of techniques to identify and analyze packed code. More details on Mandiant's tool will be presented at Blackhat DC 2007 and published at http://www.mandiant.com.

Strings

To understand what a program does, it would be ideal to have access to an instruction manual that walks step-by-step through each of the program's functions and options. Of course, malicious programs usually don't come with instruction manuals—but you might be surprised by how much can be learned about a program simply by analyzing strings of

[4] http://peid.has.it
[5] At the time this paper was written, it was internally named Caprica6, but by the time you are reading this, the name will likely have changed.

readable text that are embedded within the program. For example, programs often print output to the screen to provide the user a status update, or to indicate that an error has occurred. These status strings and error strings end up embedded in the program's executable file and can be incredibly useful in analyzing malware.

Embedded strings can be extracted from executable files using a wide variety of tools, including Strings from Sysinternals, Bintext from Foundstone, and Hex Workshop. Whichever tool you use, be sure that it can extract strings that are represented in both ASCII and Unicode formats.

Once you have extracted strings from an executable file, pop some of the more interesting looking strings into a search engine and see what pops up. Be careful, as the information embedded in the executable could easily be inserted deliberately to mislead you or cause you to trigger a sort of reverse-honeypot.

Inside the PE File Format

PE stands for "portable executable" and is the format used by executable files on Windows systems. There is a wealth of useful information that can be extracted by examining the metadata of a PE formatted file, including:

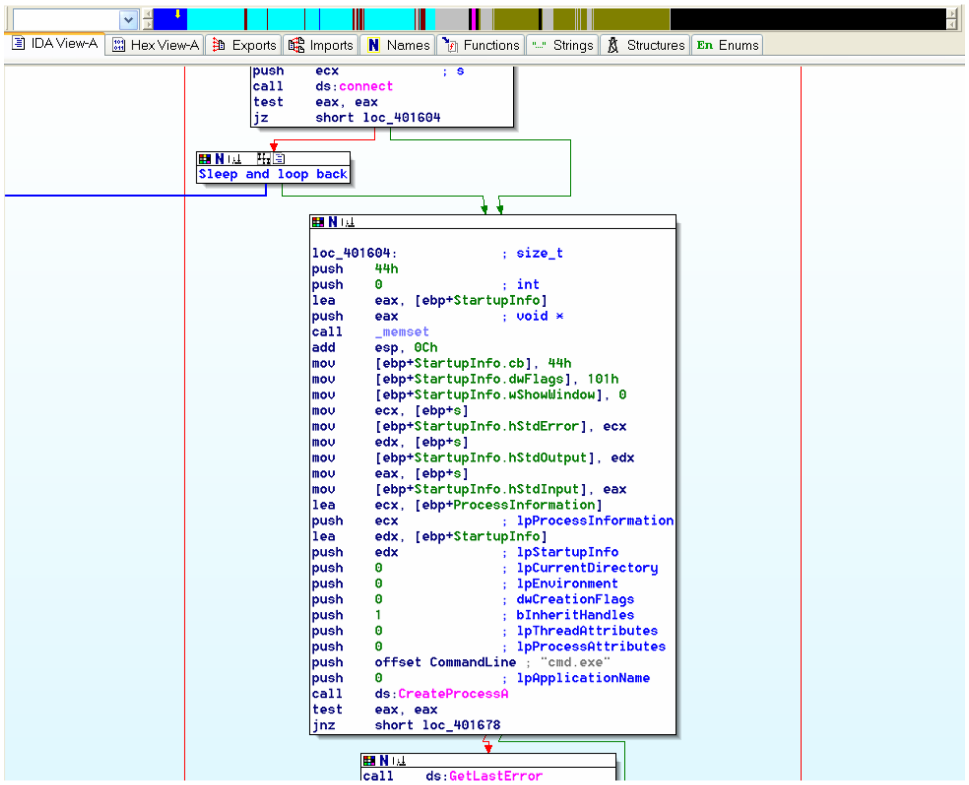
- Date and time of compilation
- Functions imported by the program
- Functions exported by the program
- Icons, Menus, Version Info, and Strings embedded in resources

There is a wide-variety of tools available that will parse PE files and allow you to extract these important details, including:

• PEview	(Wayne Radburn, http://www.magma.ca/~wjr/)
• Depends	(Steve Miller, http://www.dependencywalker.com
• PEBrowse Pro	(Russ Osterlund, http://www.smidgeonsoft.com)
• Objdump	(Cygwin, http://www.cygwin.com)
• Resource Hacker	(Angus Johnson, http://www.angusj.com/resourcehacker/)

Disassembly

After you have conducted the analysis described so far, the next step is usually to disassemble the file and analyze the assembly code instructions that make up the program. Although there are many programs that can reverse machine code to assembly language, everyone uses IDA Pro. If you are doing any serious malware analysis or other reverse engineering you need to buy a copy of IDA—it is worth it. Examining a program in IDA Pro can be somewhat intimidating at first, and certainly requires more specialized knowledge than any of the other techniques presented here. However, there is no combination of tools more powerful for malware analysis than a good disassembler (like IDA Pro) and a good debugger (like Ollydbg).



```
push    ecx                 ; s
call    ds:connect
test    eax, eax
jz      short loc_401604
```

```
Sleep and loop back
```

```
loc_401604:                 ; size_t
push    44h
push    0                   ; int
lea     eax, [ebp+StartupInfo]
push    eax                 ; void *
call    _memset
add     esp, 0Ch
mov     [ebp+StartupInfo.cb], 44h
mov     [ebp+StartupInfo.dwFlags], 101h
mov     [ebp+StartupInfo.wShowWindow], 0
mov     ecx, [ebp+s]
mov     [ebp+StartupInfo.hStdError], ecx
mov     edx, [ebp+s]
mov     [ebp+StartupInfo.hStdOutput], edx
mov     eax, [ebp+s]
mov     [ebp+StartupInfo.hStdInput], eax
lea     ecx, [ebp+ProcessInformation]
push    ecx                 ; lpProcessInformation
lea     edx, [ebp+StartupInfo]
push    edx                 ; lpStartupInfo
push    0                   ; lpCurrentDirectory
push    0                   ; lpEnvironment
push    0                   ; dwCreationFlags
push    1                   ; bInheritHandles
push    0                   ; lpThreadAttributes
push    0                   ; lpProcessAttributes
push    offset CommandLine ; "cmd.exe"
push    0                   ; lpApplicationName
call    ds:CreateProcessA
test    eax, eax
jnz     short loc_401678
```

```
call    ds:GetLastError
```

Figure 4: Disassembly of a backdoor program in IDA Pro

DYNAMIC ANALYSIS TECHNIQUES

The previous section focused on techniques that can be used to analyze malware without running it. When performing static analysis, you are in essence conducting an autopsy of the code—examining it at rest, in a dead state. When performing dynamic analysis, you actually run the malware and observe its actions. Is essence, you will create a fishbowl for the malware and then watch what it does.

In the discussion on static analysis, it was not yet important to create a safe analysis environment. Code that isn't running isn't really all that dangerous (unless it contains an exploit for IDA Pro's analysis engine[6]). However, dynamic analysis of malware must be performed in an environment that you are willing to sacrifice, and that is logically partitioned from other hosts on your network (and, hopefully, the rest of the world).

You can develop a fairly good picture of the behavior of a Windows program by simply monitoring its interaction with the file system, the registry, other processes, and the network. Although there is no single system monitoring tool that captures all of this information, you can come close with two free tools—Process Monitor from SysInternals and the open-source Wireshark. One of the interesting things about these tools is that they monitor the behavior of a whole machine rather than the behavior of the single malicious program. Therefore, it is important to be able to "filter out" normal background activity and other actions that are not attributable to the malware you are examining. Both Process Monitor and Wireshark provide sophisticated filtering capabilities. As you gain experience in malware analysis, you will also develop a "cognitive filter" based on your intuition for determining what behavior is "normal" and what is malicious. Process Monitor and

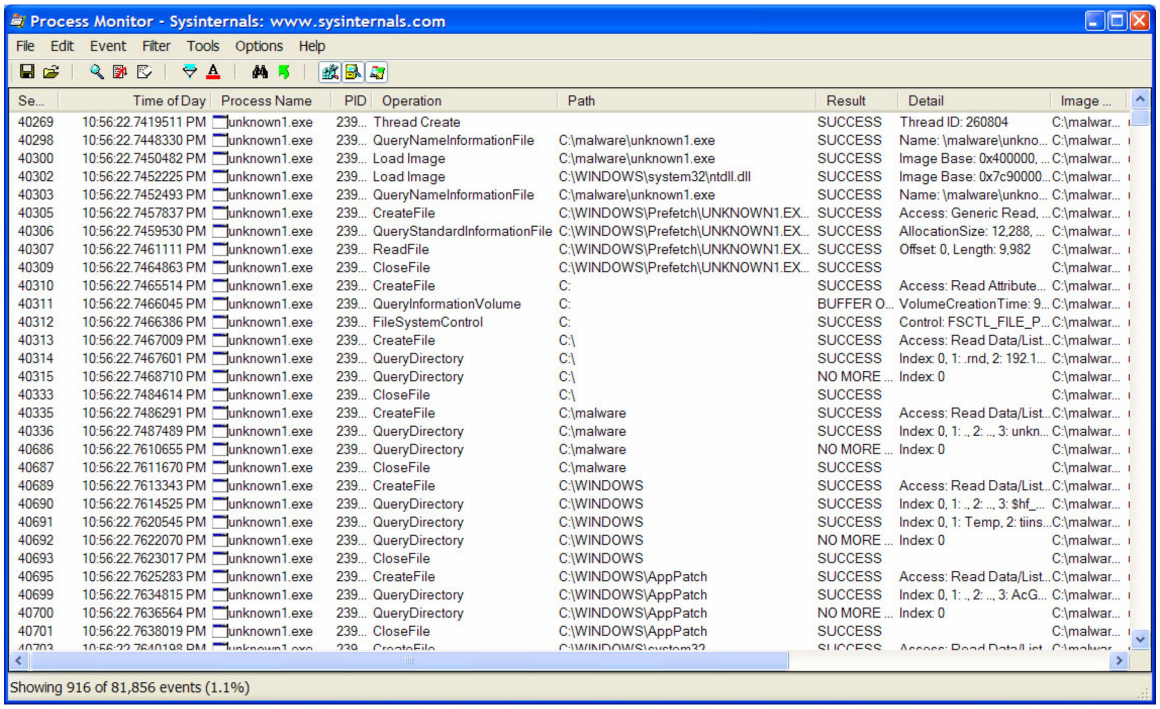
[6] http://www.securiteam.com/securitynews/5FP0G20F5U.html

Wireshark cannot automatically differentiate "good" activity from "bad" activity, or background noise from the data that is truly relevant. The tools simply collect the raw data; it is your job to interpret this data and use it to gain an understanding of the program being examined.

Process Monitor

Process Monitor is a SysInternals tool that allows users to monitor all file, registry, and process activity on Windows systems. Process Monitor works by installing a device driver that captures information about activity happening inside the kernel of the system being monitored. Although this data is captured using a device driver, the captured information is transferred to userland and presented within a simple and easy to use graphical user interface. In my opinion, Process Monitor is virtually unchallenged as the best and most powerful tool for monitoring system activity on Windows systems.

As Process Monitor captures activity, each file, registry or process operation creates a line of output in the Process Monitor window (Figure 5).

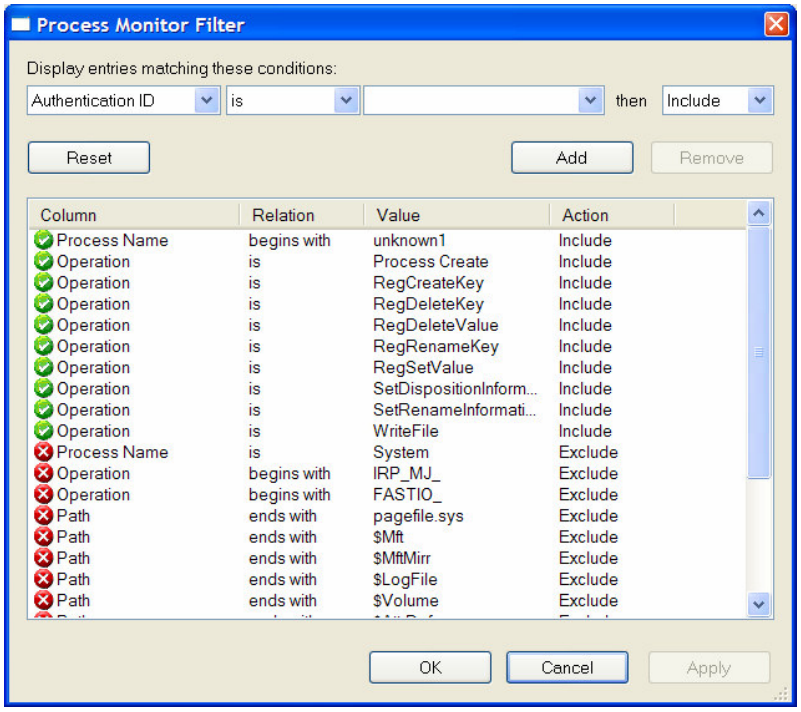


Se...	Time of Day	Process Name	PID	Operation	Path	Result	Detail	Image ...
40269	10:56:22.7419511 PM	unknown1.exe	239...	Thread Create		SUCCESS	Thread ID: 260804	C:\malwar...
40298	10:56:22.7448330 PM	unknown1.exe	239...	QueryNameInformationFile	C:\malware\unknown1.exe	SUCCESS	Name: \malware\unkno...	C:\malwar...
40300	10:56:22.7450482 PM	unknown1.exe	239...	Load Image	C:\malware\unknown1.exe	SUCCESS	Image Base: 0x400000, ...	C:\malwar...
40302	10:56:22.7452225 PM	unknown1.exe	239...	Load Image	C:\WINDOWS\system32\ntdll.dll	SUCCESS	Image Base: 0x7c90000...	C:\malwar...
40303	10:56:22.7452493 PM	unknown1.exe	239...	QueryNameInformationFile	C:\malware\unknown1.exe	SUCCESS	Name: \malware\unkno...	C:\malwar...
40305	10:56:22.7457837 PM	unknown1.exe	239...	CreateFile	C:\WINDOWS\Prefetch\UNKNOWN1.EX...	SUCCESS	Access: Generic Read, ...	C:\malwar...
40306	10:56:22.7459530 PM	unknown1.exe	239...	QueryStandardInformationFile	C:\WINDOWS\Prefetch\UNKNOWN1.EX...	SUCCESS	AllocationSize: 12,288, ...	C:\malwar...
40307	10:56:22.7461111 PM	unknown1.exe	239...	ReadFile	C:\WINDOWS\Prefetch\UNKNOWN1.EX...	SUCCESS	Offset: 0, Length: 9,982	C:\malwar...
40309	10:56:22.7464863 PM	unknown1.exe	239...	CloseFile	C:\WINDOWS\Prefetch\UNKNOWN1.EX...	SUCCESS		C:\malwar...
40310	10:56:22.7465514 PM	unknown1.exe	239...	CreateFile	C:	SUCCESS	Access: Read Attribute...	C:\malwar...
40311	10:56:22.7466045 PM	unknown1.exe	239...	QueryInformationVolume	C:	BUFFER O...	VolumeCreationTime: 9...	C:\malwar...
40312	10:56:22.7466386 PM	unknown1.exe	239...	FileSystemControl	C:	SUCCESS	Control: FSCTL_FILE_P...	C:\malwar...
40313	10:56:22.7467009 PM	unknown1.exe	239...	CreateFile	C:\	SUCCESS	Access: Read Data/List...	C:\malwar...
40314	10:56:22.7467601 PM	unknown1.exe	239...	QueryDirectory	C:\	SUCCESS	Index: 0, 1: .rnd, 2: 192.1...	C:\malwar...
40315	10:56:22.7468710 PM	unknown1.exe	239...	QueryDirectory	C:\	NO MORE ...	Index: 0	C:\malwar...
40333	10:56:22.7484614 PM	unknown1.exe	239...	CloseFile	C:\	SUCCESS		C:\malwar...
40335	10:56:22.7486291 PM	unknown1.exe	239...	CreateFile	C:\malware	SUCCESS	Access: Read Data/List...	C:\malwar...
40336	10:56:22.7487489 PM	unknown1.exe	239...	QueryDirectory	C:\malware	SUCCESS	Index: 0, 1: ., 2: .., 3: unkn...	C:\malwar...
40686	10:56:22.7610655 PM	unknown1.exe	239...	QueryDirectory	C:\malware	NO MORE ...	Index: 0	C:\malwar...
40687	10:56:22.7611670 PM	unknown1.exe	239...	CloseFile	C:\malware	SUCCESS		C:\malwar...
40689	10:56:22.7613343 PM	unknown1.exe	239...	CreateFile	C:\WINDOWS	SUCCESS	Access: Read Data/List...	C:\malwar...
40690	10:56:22.7614525 PM	unknown1.exe	239...	QueryDirectory	C:\WINDOWS	SUCCESS	Index: 0, 1: ., 2: .., 3: $hf_...	C:\malwar...
40691	10:56:22.7620545 PM	unknown1.exe	239...	QueryDirectory	C:\WINDOWS	SUCCESS	Index: 0, 1: Temp, 2: tiins...	C:\malwar...
40692	10:56:22.7622070 PM	unknown1.exe	239...	QueryDirectory	C:\WINDOWS	NO MORE ...	Index: 0	C:\malwar...
40693	10:56:22.7623017 PM	unknown1.exe	239...	CloseFile	C:\WINDOWS	SUCCESS		C:\malwar...
40695	10:56:22.7625283 PM	unknown1.exe	239...	CreateFile	C:\WINDOWS\AppPatch	SUCCESS	Access: Read Data/List...	C:\malwar...
40699	10:56:22.7634815 PM	unknown1.exe	239...	QueryDirectory	C:\WINDOWS\AppPatch	SUCCESS	Index: 0, 1: ., 2: .., 3: AcG...	C:\malwar...
40700	10:56:22.7636564 PM	unknown1.exe	239...	QueryDirectory	C:\WINDOWS\AppPatch	NO MORE ...	Index: 0	C:\malwar...
40701	10:56:22.7638019 PM	unknown1.exe	239...	CloseFile	C:\WINDOWS\AppPatch	SUCCESS		C:\malwar...

Showing 916 of 81,856 events (1.1%)

Figure 5: Process Monitor

When running Process Monitor, immediately notice that even when idle, the typical Windows system creates a LOT of events. Therefore, the key to using Process Monitor effectively is setting up accurate filters that capture the information you are interested in without missing important details. The filtering capability of Process Monitor is the single largest improvement over its progenitors Filemon and Regmon. In Filemon and Regmon, "capture filters" are configured based on simple string expressions with wildcards. Process Monitor captures everything, but then creates "display filters" containing precise compound expressions. An example of an effective filter for malware analysis is shown in Figure 6. This expression focuses on events created by a single process (named "unknown1") and focuses on events that result in some permanent change to the system, such as data written to a file or creation of a new registry value. Once you design a filter expression you like, it can be saved for future re-use.



Column	Relation	Value	Action
✓ Process Name	begins with	unknown1	Include
✓ Operation	is	Process Create	Include
✓ Operation	is	RegCreateKey	Include
✓ Operation	is	RegDeleteKey	Include
✓ Operation	is	RegDeleteValue	Include
✓ Operation	is	RegRenameKey	Include
✓ Operation	is	RegSetValue	Include
✓ Operation	is	SetDispositionInform...	Include
✓ Operation	is	SetRenameInformati...	Include
✓ Operation	is	WriteFile	Include
✗ Process Name	is	System	Exclude
✗ Operation	begins with	IRP_MJ_	Exclude
✗ Operation	begins with	FASTIO_	Exclude
✗ Path	ends with	pagefile.sys	Exclude
✗ Path	ends with	$Mft	Exclude
✗ Path	ends with	$MftMirr	Exclude
✗ Path	ends with	$LogFile	Exclude
✗ Path	ends with	$Volume	Exclude

Figure 6: Process Monitor filter

Wireshark

Wireshark is a multi-platform, open-source network protocol analyzer that captures, analyzes, and filters network traffic. Because there are already many good free tutorials and papers on using Wireshark, I won't focus on the mechanics of using Wireshark here[7]. Wireshark is a very useful tool, but it does have some drawbacks for performing malware analysis. Most notably, Wireshark does not know what process generates each packet of captured network data, so it can be difficult to determine if a packet was generated by the malicious program you are analyzing. One alternative to Wireshark is Port Explorer from DiamondCS. Port Explorer monitors network traffic at the connection level, and unlike Wireshark records details about which process generates each connection.

DEBUGGING

In many cases, simple static analysis and dynamic analysis with Process Monitor and Wireshark will reveal the answers to the most important questions about a particular piece of malware. However, the methods presented so far are not sufficient for analyzing full-featured backdoors or botnet clients that may use custom encoding methods, complicated sets of commands, and multiple layers of obfuscated or encrypted data. The fastest way to perform full-blown analysis of these more complicated programs is to use a combination of static analysis with IDA and dynamic analysis with a good debugger like Ollydbg or Windbg.

Also, keep an eye on some of the new scriptable debugging frameworks like Paimei[8] and Vtrace[9]. These tools provide a great platform for building complicated automated analysis

[7] See http://www.wireshark.org/news/20060714.html for examples of a Wireshark tutorial
[8] http://pedram.redhive.com/PaiMei/docs/
[9] http://www.kenshoto.com/vtrace/

modules, and will likely be used in the future to automate away some of the pain of manual analysis using a typical debugger.

CONCLUSION

This paper provided a very high-level introduction to the topic of malware analysis, and some practical techniques and tools that can be used to conduct limited analysis of Windows programs of unknown functionality. The material covered here truly is the very small tip of the very large iceberg of the techniques and knowledge required to master malware analysis. If you would like to learn more, there is a wealth of additional information available in forums like www.openrce.org and www.offensivecomputing.net. For more detailed information about static analysis I highly recommend reading *Reversing: Secrets of Reverse Engineering* by Eldad Eilam. Also, take a look at Mandiant's hands-on malware analysis classes, where we cram as much malware analysis fun as can possibly fit into three days. If you have any questions about the material presented here, or have any suggestions for improvement, please contact me at kris.kendall@mandiant.com. Happy reversing!