SANS

www.sans.org

FORENSICS 508

ADVANCED DIGITAL FORENSICS AND INCIDENT RESPONSE

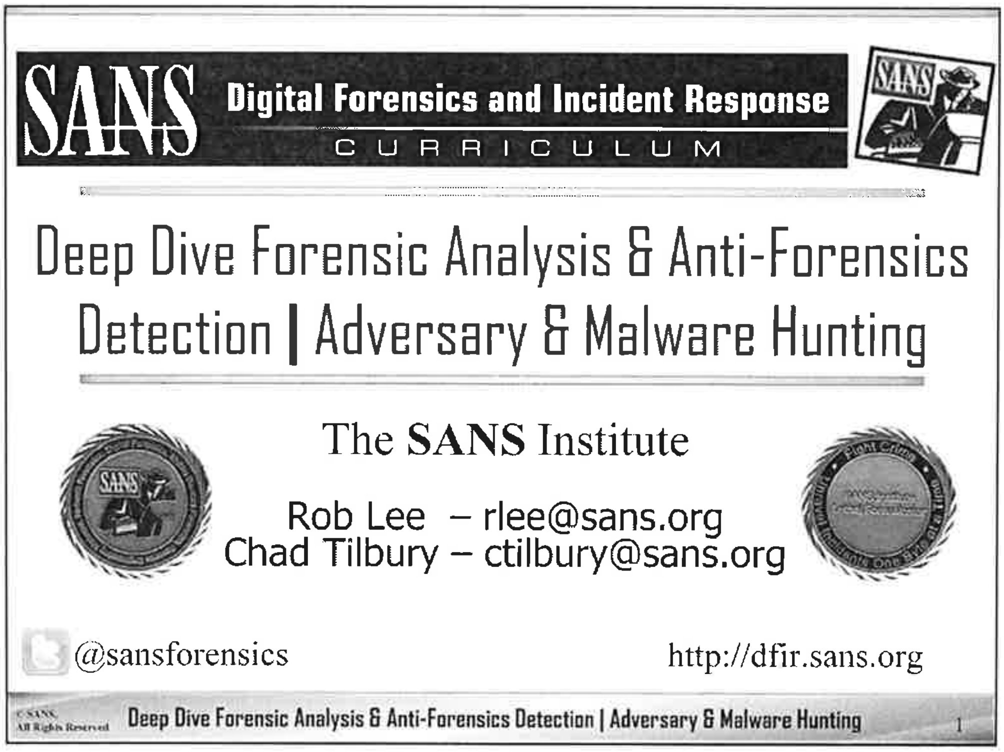
508.4
508.5

Deep Dive Forensic Analysis & Anti-Forensics Detection

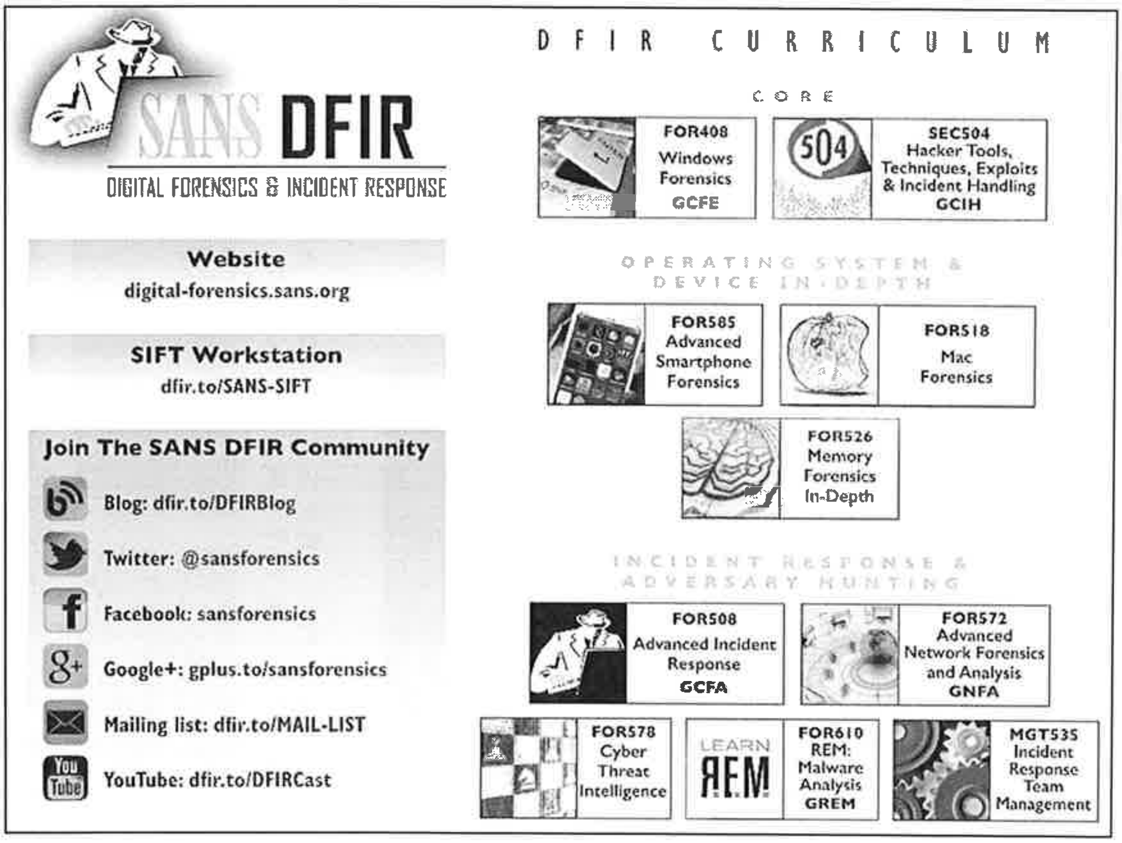
Adversary & Malware Hunting

The right security training for your staff, at the right time, in the right location.

For508_4_5_A04



Author: Rob Lee
rlee@sans.org
http://twitter.com/robtlee
http://twitter.com/sansforensics



This page intentionally left blank.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

Website

digital-forensics.sans.org

SIFT Workstation

dfir.to/SANS-SIFT

Join The SANS DFIR Community

- **Blog:** dfir.to/DFIRBlog
- **Twitter:** @sansforensics
- **Facebook:** sansforensics
- **Google+:** gplus.to/sansforensics
- **Mailing list:** dfir.to/MAIL-LIST
- **YouTube:** dfir.to/DFIRCast

DFIR CURRICULUM

CORE

FOR408
Windows Forensics
GCFE

SEC504
Hacker Tools, Techniques, Exploits & Incident Handling
GCIH

OPERATING SYSTEM & DEVICE IN-DEPTH

FOR585
Advanced Smartphone Forensics

FOR518
Mac Forensics

FOR526
Memory Forensics In-Depth

INCIDENT RESPONSE & ADVERSARY HUNTING

FOR508
Advanced Incident Response
GCFA

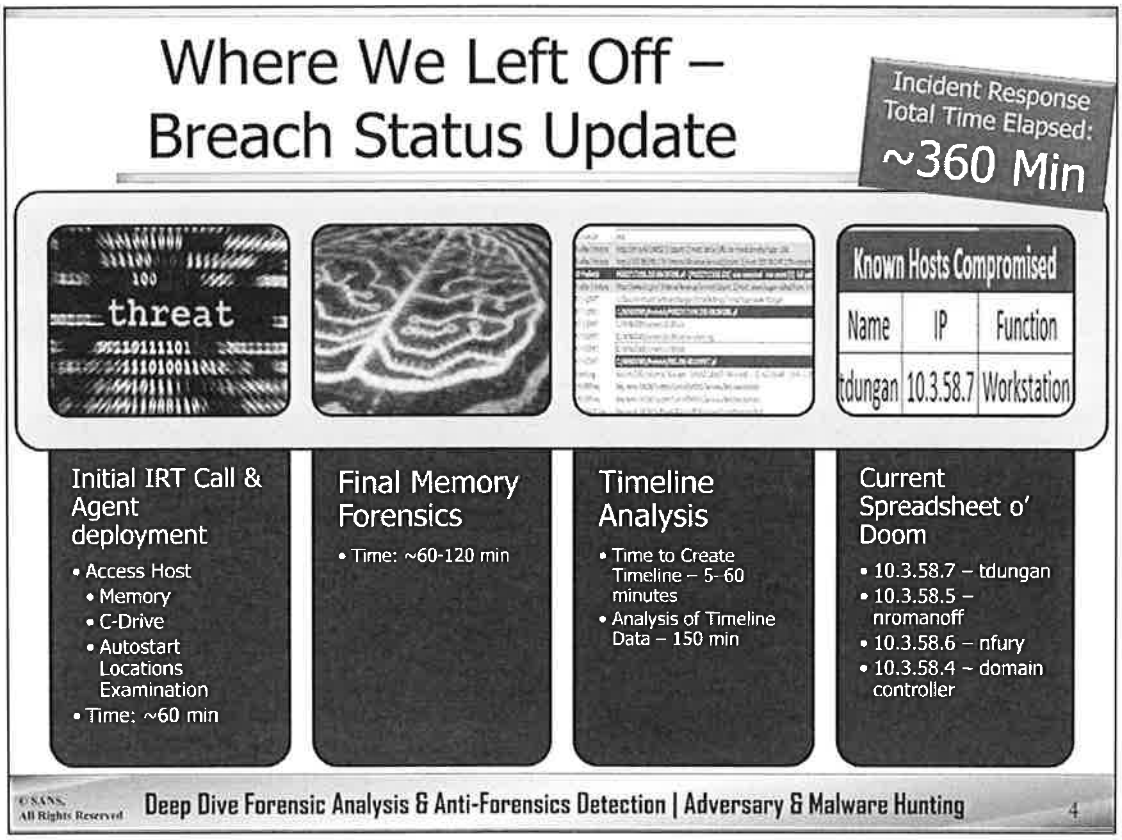
FOR572
Advanced Network Forensics and Analysis
GNFA

FOR578
Cyber Threat Intelligence

LEARN REM

FOR610
REM: Malware Analysis
GREM

MGT535
Incident Response Team Management



This page intentionally left blank.

Where We Left Off – Breach Status Update Host Indicators

Possible Evidence of Timestomping

Infector
- C:\Documents and Settings\tdungan\Local Settings\Temp\pkxezy1tji98.exe

svchost.exe
- C:\windows\system32\dllhost\winclient.reg
- C:\windows\system32\dllhost\svchost.exe
- C:\(Documents and Settings|User)\<username>\Local Settings\Temp\a.exe
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Netman\domain

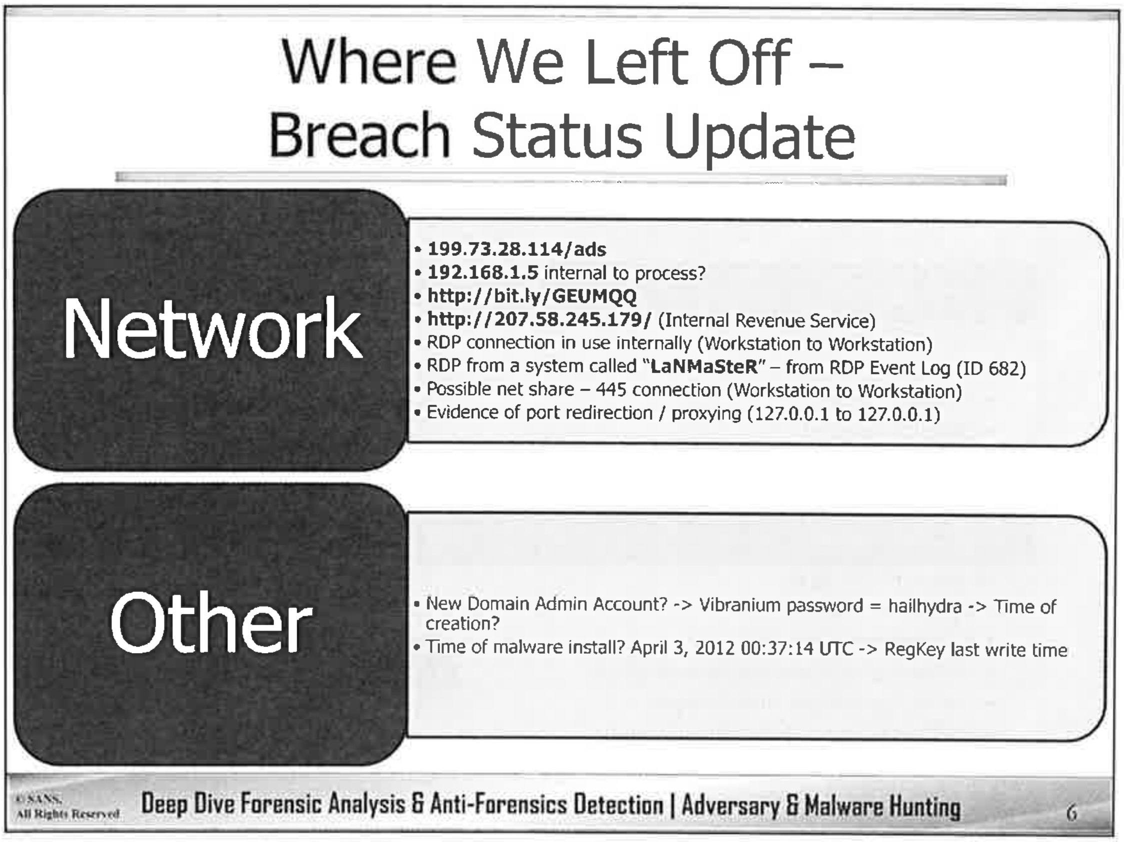
pe.exe
- C:\Windows\system32\pe.exe
- HKEY_USER/Software/Sysinternals/PsExec

spinlock.exe
- C:\windows\system32\spinlock.exe
- python25.dll
- bz2.pyd
- spinlock.manifest
- HKLM/Software/Microsoft/ESENT/Process/spinlock

hyvy.exe
- http://199.73.28.114:443/HYvy.exe (Downloaded)
- RemotePIShell.exe.manifest

hydrakatz.exe
- sekurlsa.dll

This page intentionally left blank.

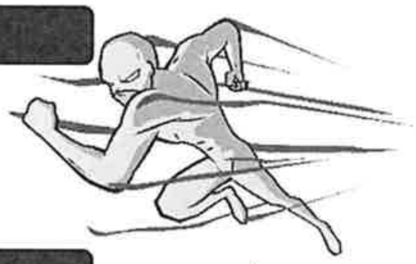


This page intentionally left blank.

Switching from Rapid Analysis to Deep Dive Analysis

Immediate/Quick Response

- Answers in 4-6 hours
  - Enterprise Response
  - Memory Forensics
  - Timeline Analysis
  - Initial Incident Assessment
  - Threat Indicators Developed



Deep Dive Analysis

- Answers in 1-2 days
  - Deep Dive Analysis
  - Anti-Forensic Detection
  - Malware and Adversary Identification
  - Threat Capabilities and Purpose
  - Additional Threat Indicators Created

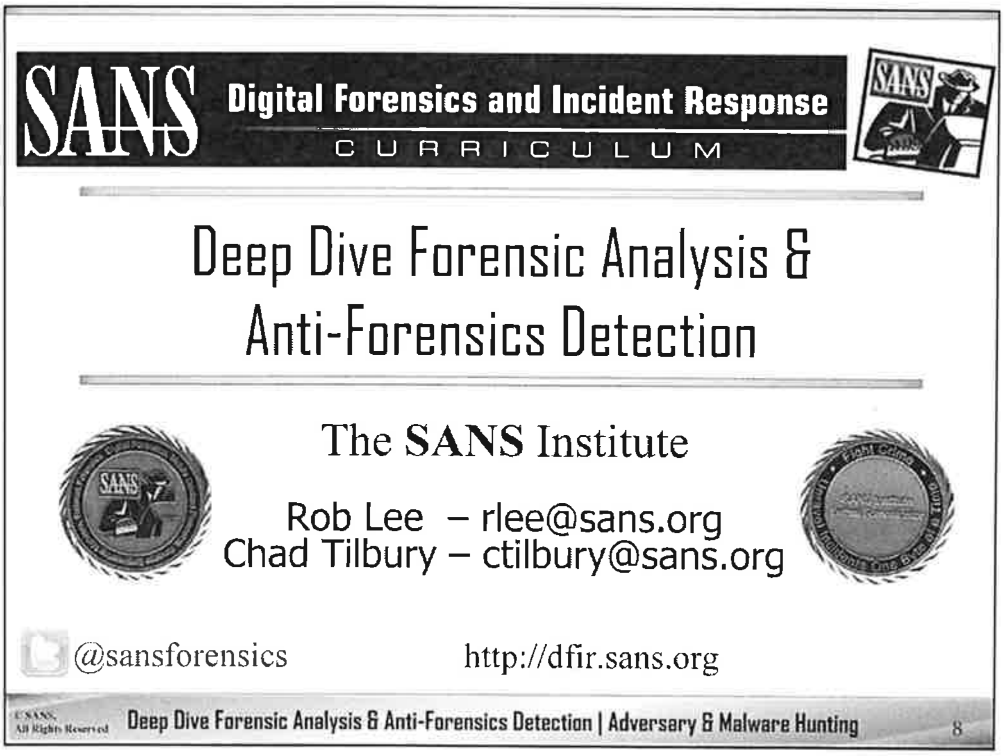
At this point, we will move from our rapid triage and analysis mode designed to produce answers in only a few hours. During the initial phase we concentrated on analytical technical that were quick and resulted in the ability to provide quick turn-around answers during incident response. During the first 24 hours, the majority of your work will likely be immediate and quick response analysis techniques.

However, not every incident can be solved with our quick analysis techniques. In cases where adversaries utilize more efficient anti-forensic techniques or are more careful about their movements around your enterprise, in order to discover their tools, techniques and procedures to pull out effective cyber threat intelligence triggers we will need to dive deeper.

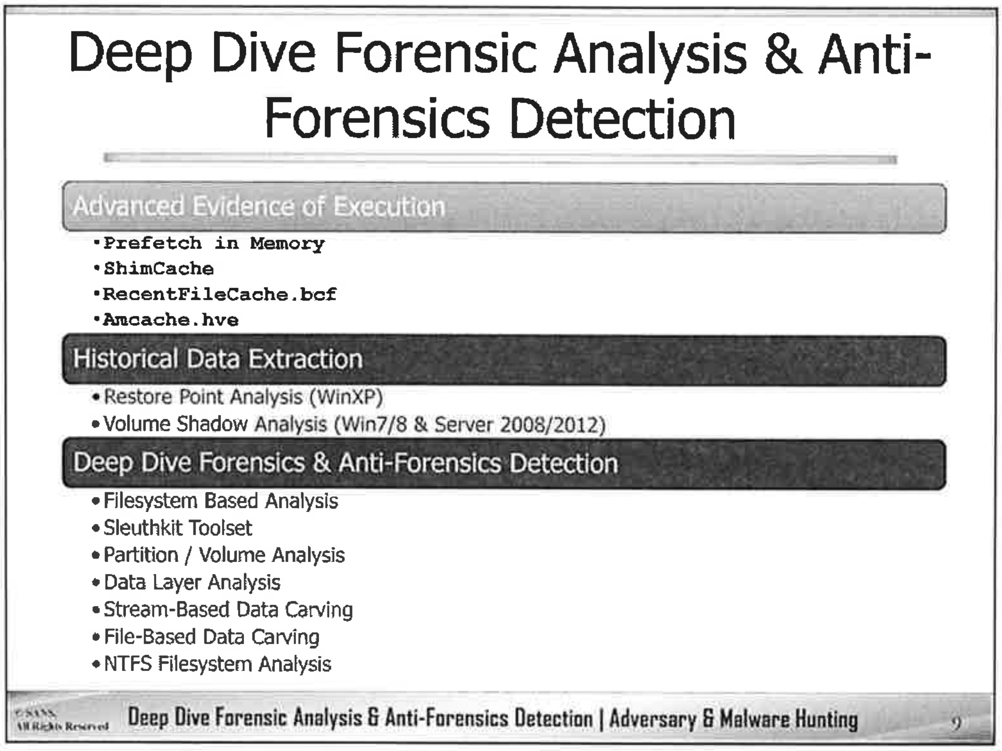
From here forward in the class till the last day are techniques that we would not recommend to accomplish on EVERY system, but systems that you feel might need a deeper look. It is hard to tell exactly which systems might need this additional scrutiny. As a result, the initial assessment might help you triage and identify systems that might yield the most data for a deep dive analysis.

Deep dive analysis will usually also give us a better look into why the threat has decided to target your organization and attempt to answer questions to their intent. "What are they after?" "How skilled is this threat group?" "Do they use anti-forensic techniques to hide?" "Are they good at hiding their malware?"

In the end, deep dive analysis techniques cannot be applied to every system, it is too time consuming and slow to achieve. But doing it on specific systems would be a wise decision.



Author: Rob Lee
rlee@sans.org
http://twitter.com/robtlee
http://twitter.com/sansforensics

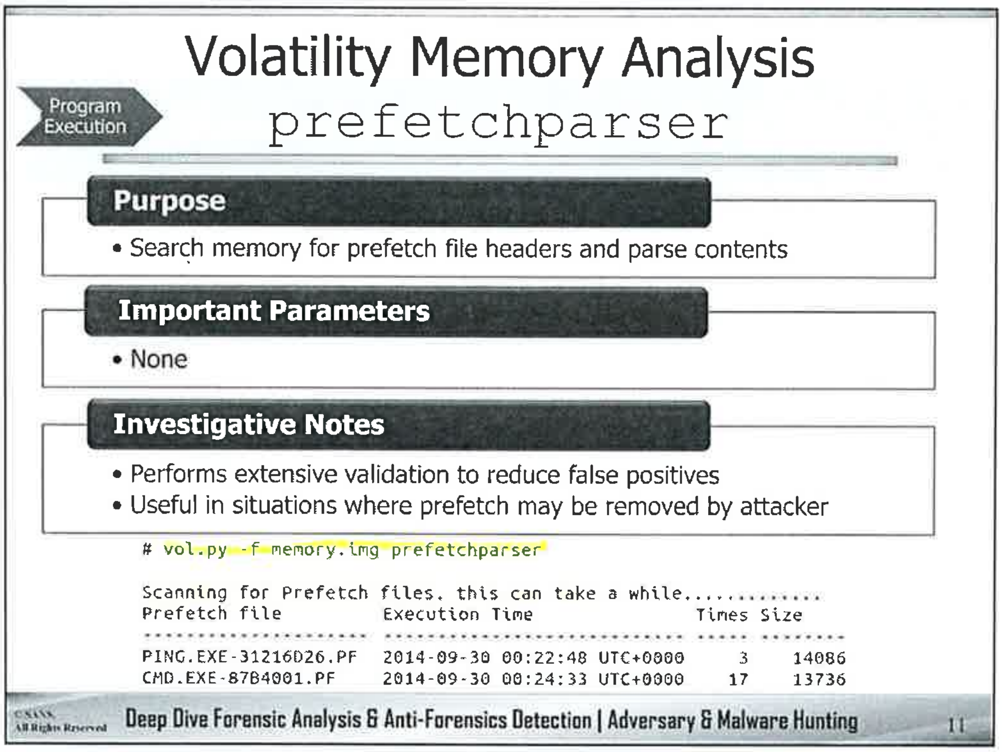


This page intentionally left blank.

SANS Digital Forensics and Incident Response
CURRICULUM

Advanced Evidence of Execution

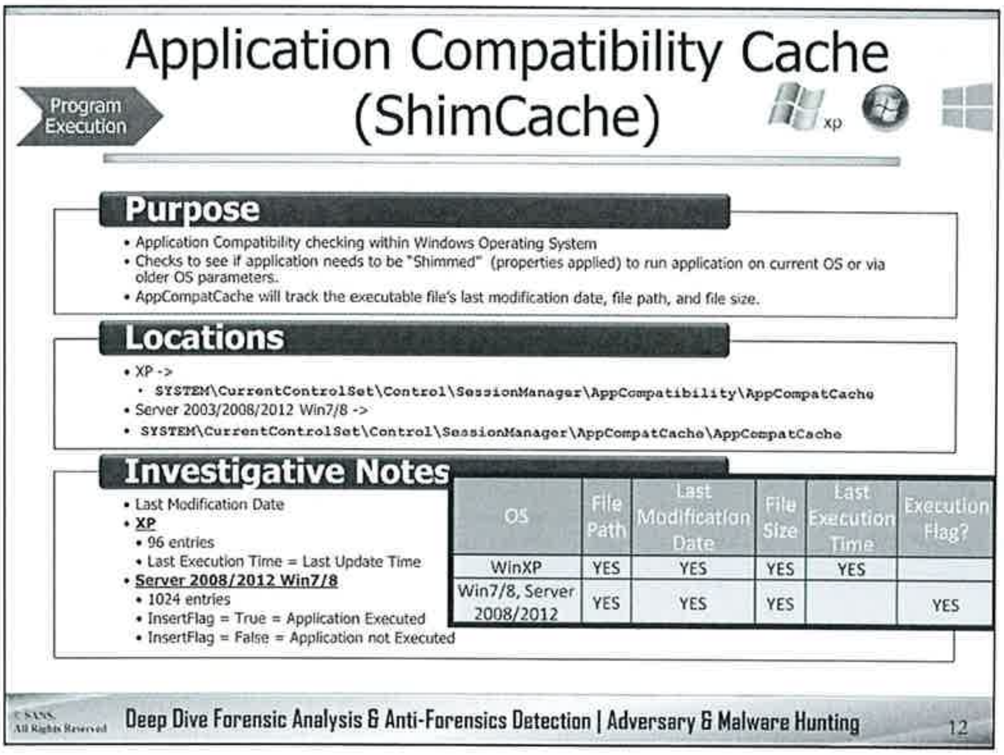
This page intentionally left blank.



Forensic examiners have long looked to prefetch for clues during intrusion investigations, and now slowly but surely our adversaries have caught up. It is not uncommon to see an attacker delete or wipe prefetch files upon leaving a machine. This (relatively) new behavior requires new artifacts and new tools that can dig deeper.

The **prefetchparser** for the Volatility memory analysis framework is a specialized plugin designed to comb memory address space for prefetch files. [1] The plugin was written by Dave Lassalle and does an excellent job of recovering prefetch files that may no longer be present on disk. Instead of attempting to recover prefetch from the Windows cache manager like the **dumpfiles** plugin, **prefetchparser** takes the more rigorous approach of scanning virtual memory for prefetch headers and then performing strong validation to reduce false positives. It then parses the internal metadata for each file and presents it to the analyst. It is an excellent choice for your toolbox when you encounter more advanced adversaries.

```
root@siftworkstation:/# vol.py -f memory shimcache | less
2009-07-14 01:14:21 UTC+0000   \??\C:\Windows\system32\ipconfig.exe
2012-04-03 21:19:52 UTC+0000   \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:18:14 UTC+0000   \??\C:\Windows\oSCMpGpk.exe
2012-04-03 21:14:56 UTC+0000   \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:13:29 UTC+0000   \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:13:07 UTC+0000   \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:12:04 UTC+0000   \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:11:07 UTC+0000   \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:03:23 UTC+0000   \??\C:\Windows\TopLZAGU.exe
```



The Application Compatibility Cache was designed by Microsoft to detect program compatibly challenges when a program launches. A program might be built to work for another version of Windows, so to avoid compatibility issues, Microsoft created a subsystem that would allow a program to invoke properties of different operating system versions and environment changes.

The compatibility capability is seen by regular users when they execute a program through the compatibility wizard or when they set an application's capabilities to run in "WinXP" mode. The different modes are essentially called "Shims." By default, there are 100s of shims that exist on a standard Windows installation. Windows looks at the AppCompatCache registry key to figure out if a program needs shimming for compatibility. [1]

The most useful pieces of this to the responder is that the AppCompatCache will track the executable file's last modification date, file path, and file size. On Windows XP (32 bit), it was also track the last time executed (last update time.) The reason this key is useful to examiners is that it is one step closer to showing if an application has possibly been executed and could aid an incident responder look for malware executed on the system.

When a program is shimmed, registry entries are created to notify the system. As the program (.exe, .bat, or .dll) loads, it will be checked in the Shim database and perform lookups in the system compatibility database. One of the more interesting aspects of the AppCompatCache is that each application is checked regardless if it has been shimmed or not.

The registry key paths for the AppCompatCache are:

<u>XP</u>

- **SYSTEM\CurrentControlSet\Control\SessionManager\AppCompatibility\AppCompatCache**
- 96 entries

Server 2003/2008/2012 Win7/8

- **SYSTEM\CurrentControlSet\Control\SessionManager\AppCompatCache\AppCompatCache**
- Server 2003 = 512 entries
- Win7/8, Server 2008/2012 = 1024 entries

The AppCompatCache is implemented in one of four ways depending on the operating system. This also ends up changing the type of data stored in each of the registry entries tied to the AppCompatCache entries. The most notable difference occurs in Windows XP which lists the last execution time of a program in addition to file path, file size, and the application's last modification time. In Win2003, Server 2008/2012, and Win7/8 the registry location stores the following useful information: file path, file size, and applications last modification time. The last execution time does not exist in the latter operating systems. [2] The cache does not exist on Windows 2000 systems or older. [3]

OS	File Path	Last Modification Date	File Size	Last Execution Time	Execution Flag?
WinXP	YES	YES	YES	YES	
2003 & XP 64bit	YES	YES	YES		YES
Win7/8, Server 2008/2012	YES	YES	YES		YES

The registry for the AppCompatCache entries are only written on system shutdown and not before.

When reviewing the output from the AppCompatCache it is good to note the following:

1. Most recent events are on top
2. New entries are written on shutdown

Some of the most useful capabilities of the AppCompatCache is the fact that if an attacker has removed the malware from the system and was careful to also delete the prefect (.pf) file as well then the AppCompatCache entries might provide some of the only clues that the application existed in the first place.

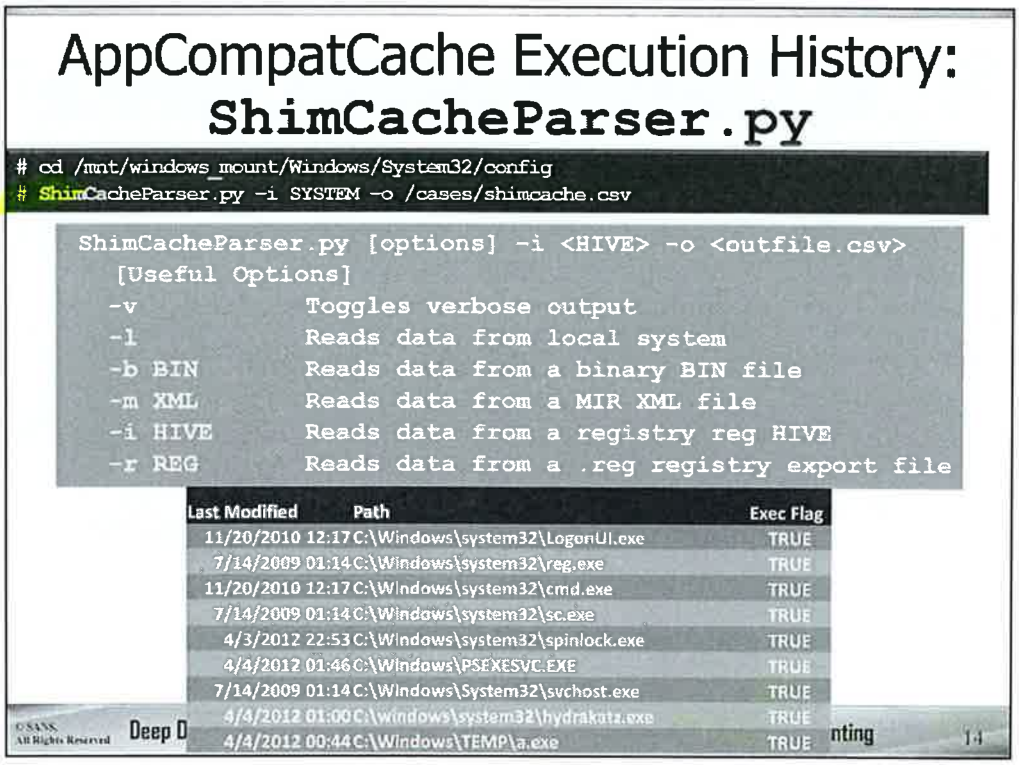
Starting with VISTA+, the existence of an entry in the AppCompatCache registry key does not prove execution, however it is very likely. In VISTA+ there is an existence of a flag called the "InsertFlag" in the structure of the AppCompatCache Entry structure (VISTA+ systems only) that shows, on initial testing, that if it is set indicates that the program was executed on the system. If the flag is not set, the application did not execute. [2][3]

[1] - Secrets of the Application Compatibility Database – http://www.alex-ionescu.com/?p=39

[2] - Leveraging the Application Compatibility Cache in Forensic Investigations – https://dl.mandiant.com/EE/library/Whitepaper_ShimCacheParser.pdf

[3] – Johnny AppCompatCache https://digital-forensics.sans.org/summit-archives/DFIR_Summit/Johnny-AppCompatCache-the-Ring-of-Malware-Brice-Daniels-and-Mary-Singh.pdf

[4] – Understanding Shims - http://technet.microsoft.com/en-us/library/dd837644(v=ws.10).aspx



ShimCacheParser.py is a tool for reading the Application Compatibility Shim Cache stored in the Windows registry. Metadata of files that are executed on a Windows system are placed within this data structure on the running system. Upon system shutdown, this data structure is serialized to the registry in one of two registry paths depending on the operating system version.

The format of this data, as well as the types of information stored also vary between operating system which is summarized below:

- Windows XP 32-bit: File Path, $STANDARD_INFORMATION Last Modified Time, File Size, and Last Update Time
- Windows 2003 and XP 64-bit: File Path, $STANDARD_INFORMATION Last Modified Time, and File Size
- Windows Vista and later: File Path, $STANDARD_INFORMATION Last Modified Time, Shim Flags

The script will find these registry paths, automatically determine their format, and return the data in an optional CSV format. During testing it was discovered that on Windows Vista and later, files may be added to this cache if they were browsed to by explorer.exe and never actually executed. When these same files were executed, the 2nd least significant bit in the flags field was set. During testing it was possible to identify if processes were executed based on this flag being set. This flag's true purpose is currently unknown and is still being testing for consistency, so it should not be currently used to definitively conclude that a file may or may not have executed.

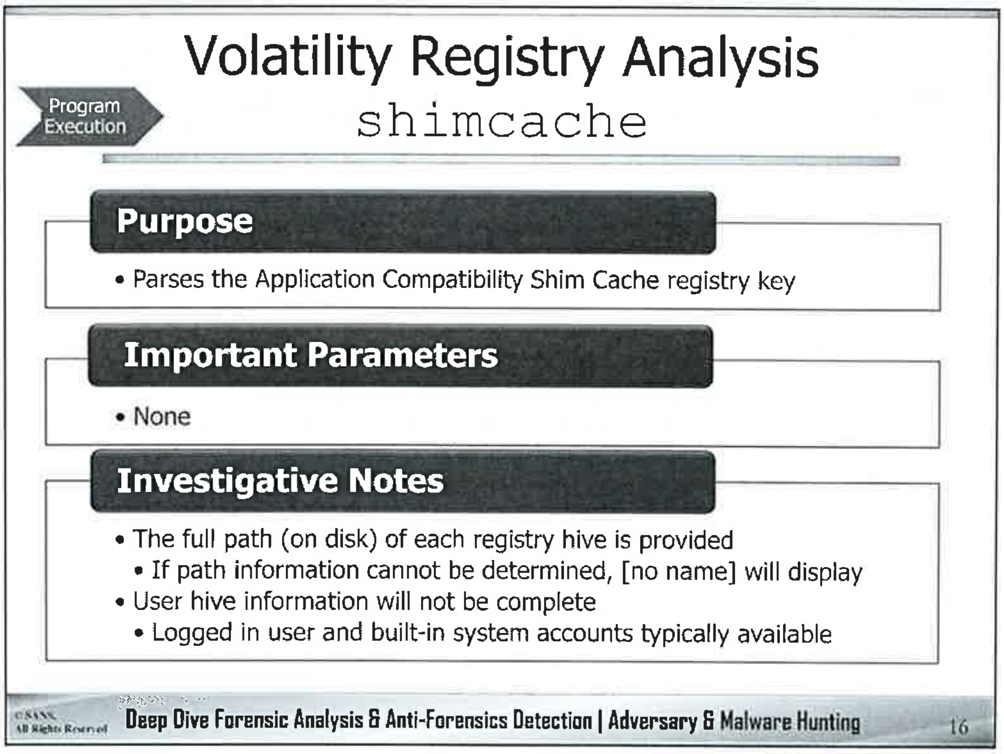
The output CSV file is set with the (-o, --output) argument. If no output file is specified, the data will be printed to STDOUT. ShimCacheParser will search each ControlSet and will only return unique entries by default. If you want to display duplicates as well as the full registry path where the data was taken use the verbose (-v, --verbose) option.

[1] –From https://github.com/mandiant/ShimCacheParser

Last Modified	Path	Exec Flag
11/20/2010 12:17	C:\Windows\system32\LogonUI.exe	TRUE
7/14/2009 01:14	C:\Windows\system32\reg.exe	TRUE
11/20/2010 12:17	C:\Windows\system32\cmd.exe	TRUE
7/14/2009 01:14	C:\Windows\system32\sc.exe	TRUE
4/3/2012 22:53	C:\Windows\system32\spinlock.exe	TRUE
4/4/2012 01:46	C:\Windows\PSEXESVC.EXE	TRUE
7/14/2009 01:14	C:\Windows\System32\svchost.exe	TRUE
4/4/2012 01:00	C:\windows\system32\hydrakatz.exe	TRUE
4/4/2012 00:44	C:\Windows\TEMP\a.exe	TRUE

When reviewing the output from the AppCompatCache it is good to note the following:

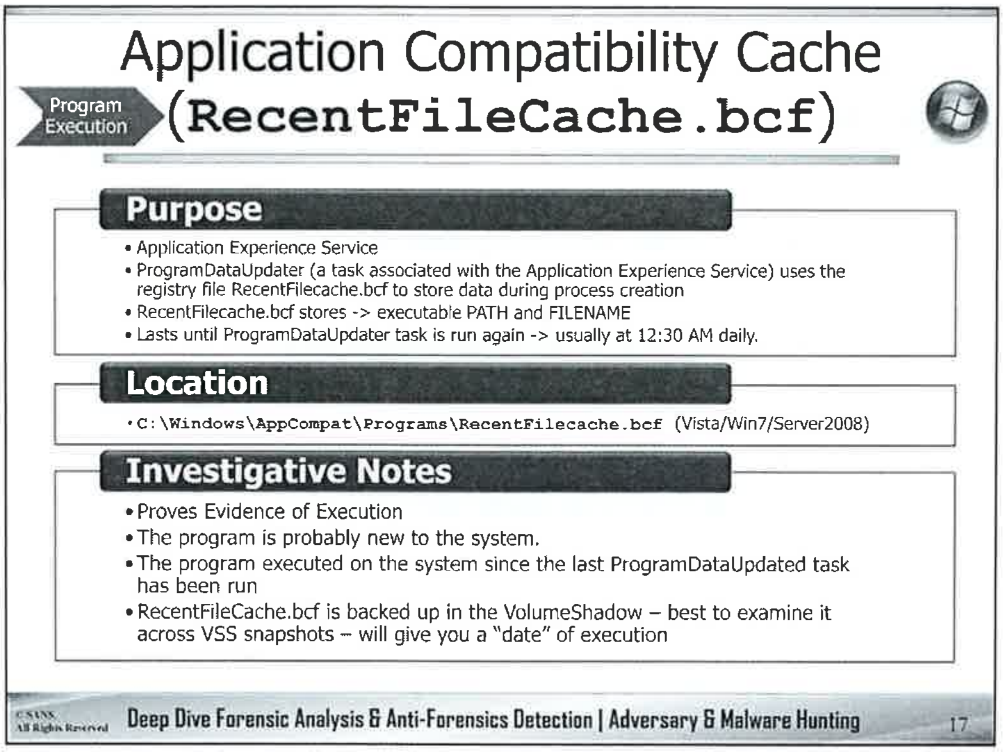
- Most recent events are on top
- New entries are written on shutdown
- If Exec Flag = True, then application was executed



In addition, the volatility plugin "**shimcache**" can also be used to extract AppCompatCache entries from a memory image from the registry hives that are loaded in RAM.

The shimcache plugin in volatility will use the AppCompatCache elements in memory to carve out recent checks done. It will be useful to compare the output of this capability against that pulled from the SYSTEM registry hive.

```
root@siftworkstation:/# vol.py -f memory shimcache | less
2009-07-14 01:14:21 UTC+0000    \??\C:\Windows\system32\ipconfig.exe
2012-04-03 21:19:52 UTC+0000    \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:18:14 UTC+0000    \??\C:\Windows\oSCMpGpk.exe
2012-04-03 21:14:56 UTC+0000    \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:13:29 UTC+0000    \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:13:07 UTC+0000    \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:12:04 UTC+0000    \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:11:07 UTC+0000    \??\C:\Windows\PSEXESVC.EXE
2012-04-03 21:03:23 UTC+0000    \??\C:\Windows\TopLZAGU.exe
```



One of the more interesting additions to the Application Compatibility Cache research for forensics is that the ShimCache database stores information for long term execution. However, the ShimCache doesn't get written until system shutdown/restart. What about recent additions? It is possible that they might exist in the RecentFileacache.bcf file.

The Application Compatibility Cache is tied to the Application Experience. It is a service that "checks a Microsoft maintained database for known problems with popular programs and automatically enables workarounds, either at first installation (using UAC) or at application launch." [1] ProgramDataUpdater (a task associated with the Application Experience Service) uses the registry file RecentFilecache.bcf during process creation.

The RecentFilecache.bcf file is found here **C:\Windows\AppCompat\Programs\RecentFilecache.bcf**

RecentFilecache.bcf file is a very small an temporary registry hive that is managed by the ProgramDataUpdater scheduled service. This service collects program telemetry information for the Microsoft Customer Experience Improvement Program.

RecentFileCache.bcf contained references to programs that were recently copied or downloaded and executed on the system. This makes it extremely useful to catch malware as they are likely to be new programs on a system. The registry hive file contains the full path and name of the program in hex and is fairly easy to dump. However, the data in the file is merely temporary. The storage of this information in the RecentFileCache.bcf file is cleared when the ProgramDataUpdater task is run each time.

The ProgramDataUpdater task is scheduled to run each day at 12:30 AM which would clear out the contents of this hive file. It really is only a temporary storage location and aptly named as the RecentFileCache.bcf. [2]

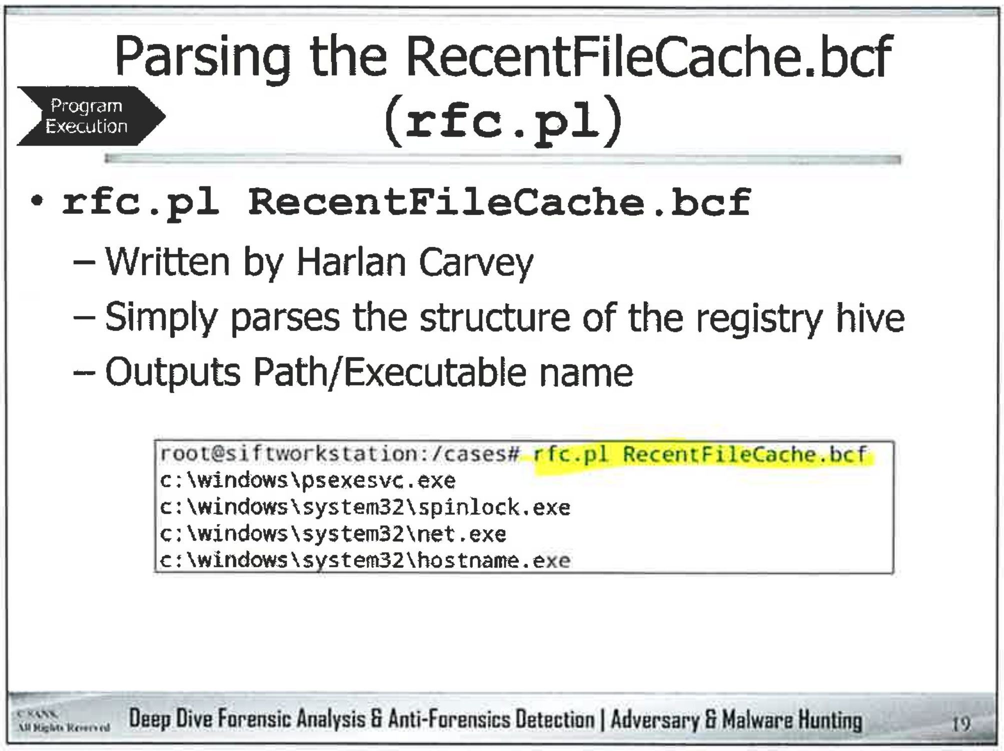
Some general rules about files found in the RecentFileCache.bcf

1. The program is probably new to the system.
2. The program executed on the system since the last ProgramDataUpdated task has been run
3. RecentFileCache.bcf is backed up in the VolumeShadow – best to examine it across VSS snapshots – will give you a "date" of execution

Harlan Carvey's release of rfc.pl (also located on the SIFT) can easily extract the data from the RecentFileCache.bcf. It doesn't show the last time of execution.

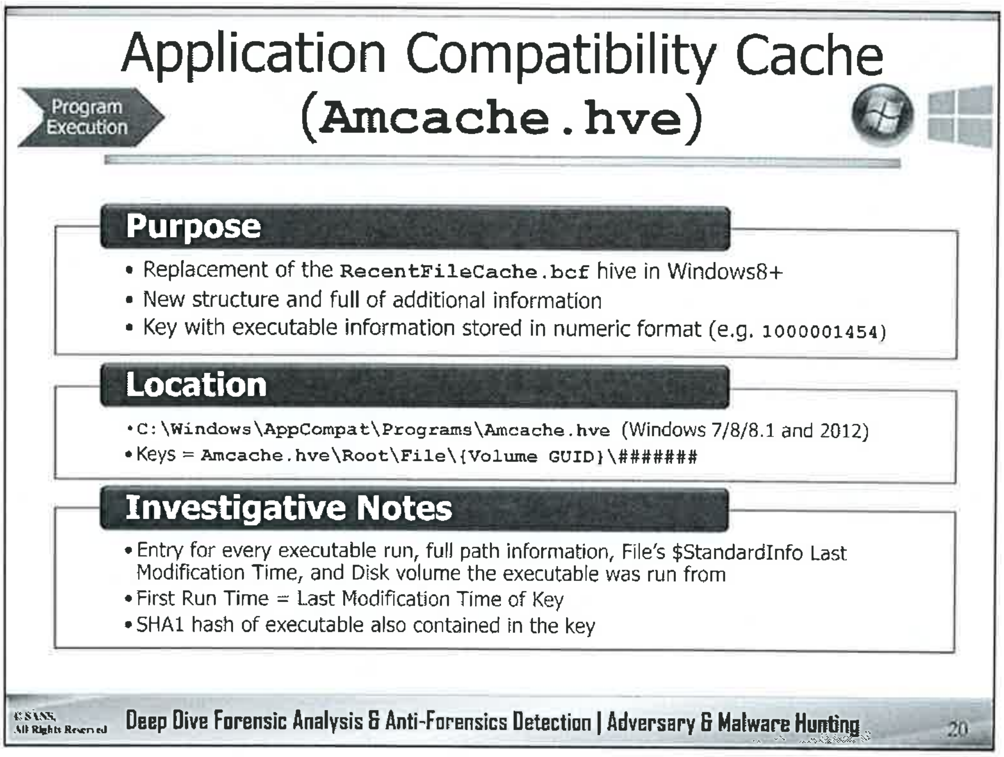
[1] - Application Experience - http://www.blackviper.com/windows-services/application-experience/

[2] - Revealing the RecentFileCache.bcf File
http://journeyintoir.blogspot.in/2013/12/revealingrecentfilecachebcf-file.html



rfc.pl is a simple program written by Harlan Carvey that is used to parse the contents of the RecentFileCache.bcf file. It will output the executable path and filename.

```
root@siftworkstation:/cases# rfc.pl RecentFileCache.bcf
c:\windows\psexesvc.exe
c:\windows\system32\spinlock.exe
c:\windows\system32\net.exe
c:\windows\system32\hostname.exe
```



Starting in Windows 8+ and in recently updated Windows 7 systems, the `RecentFileCache.bcf` hive has been replaced with a new registry hive called "`Amcache.hve`." This new hive contains an exciting amount of new information related to tracking executables, where they were executed from, and more.

The Amcache.hve will contain the executable full path, file's $StandardInfo Last Modification Time, SHA1 hash (remove leading 0000), and in some cases file information such as Version, ProductName, CompanyName, and Description.

When you open the hive in a registry viewer of choice, browse to the `Amcache.hve\Root\File\{Volume GUID}\` parent key. The Volume GUID will line up with the `MountedDevices` key found under `System\MountedDevices [1]`. This is a critical piece of information to understand as you can determine the exact volume an executable was run from using this information.

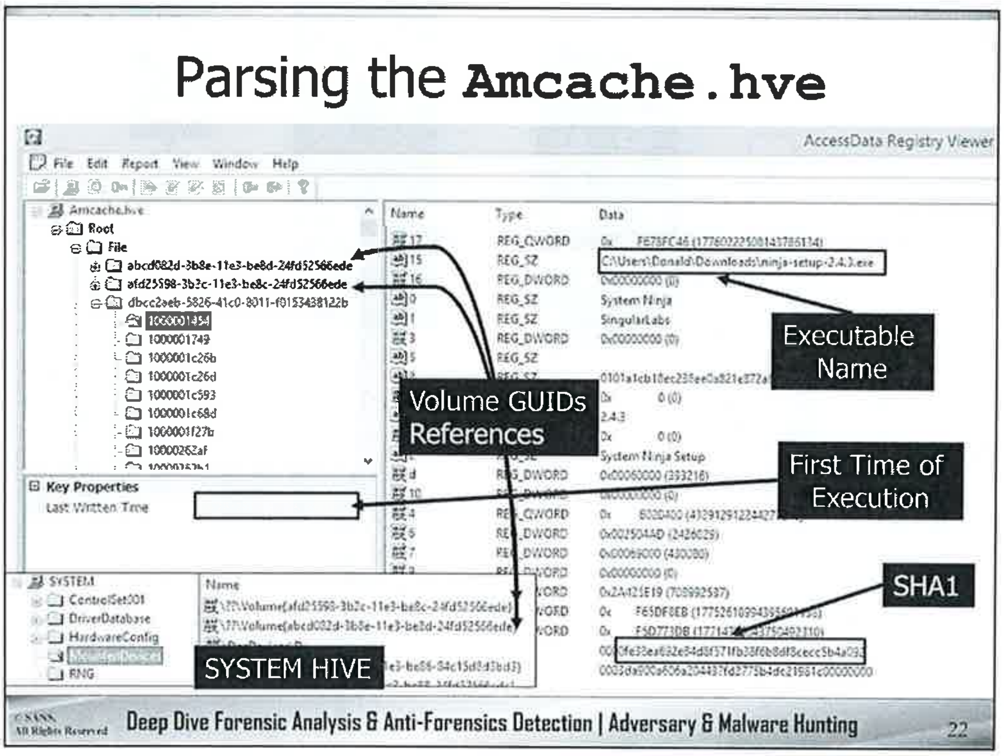
Under the `Amcache.hve\Root\File\{Volume GUID}\` key you will find numeric keys (e.g. 1000001454). Each of these keys represents a different executable file run under Windows. The last write time of the numeric key is most likely the first time a specific program was executed.

Value Names are in hexadecimal and range from 0 to 17 and then two extra entries for 100 and 101 are seen. Yogesh Khatri has reversed out these values and their meanings.[2] (Note: For Encase users Yogesh also created an EnCase EnScript to parse this key downloadable from his website.) [2]

[1] Covered in detail in FOR408 – Windows Forensics – Section 2

[2] Amcache.hve in Windows 8 - Goldmine for malware hunters - http://www.swiftforensics.com/2013/12/amcachehve-in-windows-8-goldmine-for.html

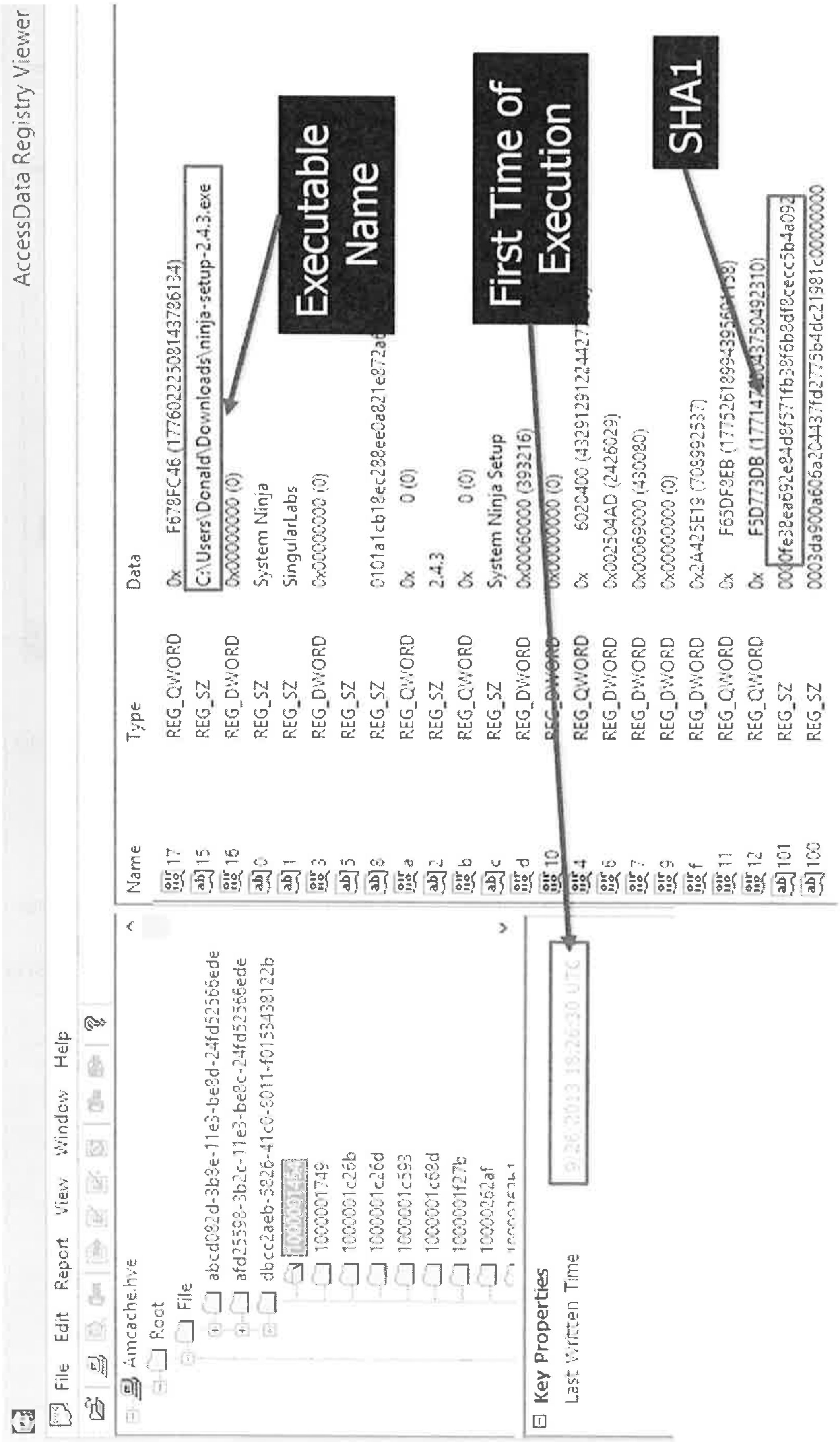
Value	Description	Data Type
0	Product Name	UNICODE string
1	Company Name	UNICODE string
2	File version number only	UNICODE string
3	Language code (1033 for en-US)	DWORD
4	SwitchBackContext	QWORD
5	File Version	UNICODE string
6	File Size (in bytes)	DWORD
7	PE Header field - SizeOfImage	DWORD
8	Hash of PE Header (unknown algorithm)	UNICODE string
9	PE Header field - Checksum	DWORD
a	Unknown	QWORD
b	Unknown	QWORD
c	File Description	UNICODE string
d	Unknown, maybe Major & Minor OS version	DWORD
f	Linker (Compile time) Timestamp	DWORD - Unix time
10	Unknown	DWORD
11	Last Modified Timestamp	FILETIME
12	Created Timestamp	FILETIME
15	Full path to file	UNICODE string
16	Unknown	DWORD
17	Last Modified Timestamp 2	FILETIME
100	Program ID	UNICODE string
101	SHA1 hash of file	UNICODE string

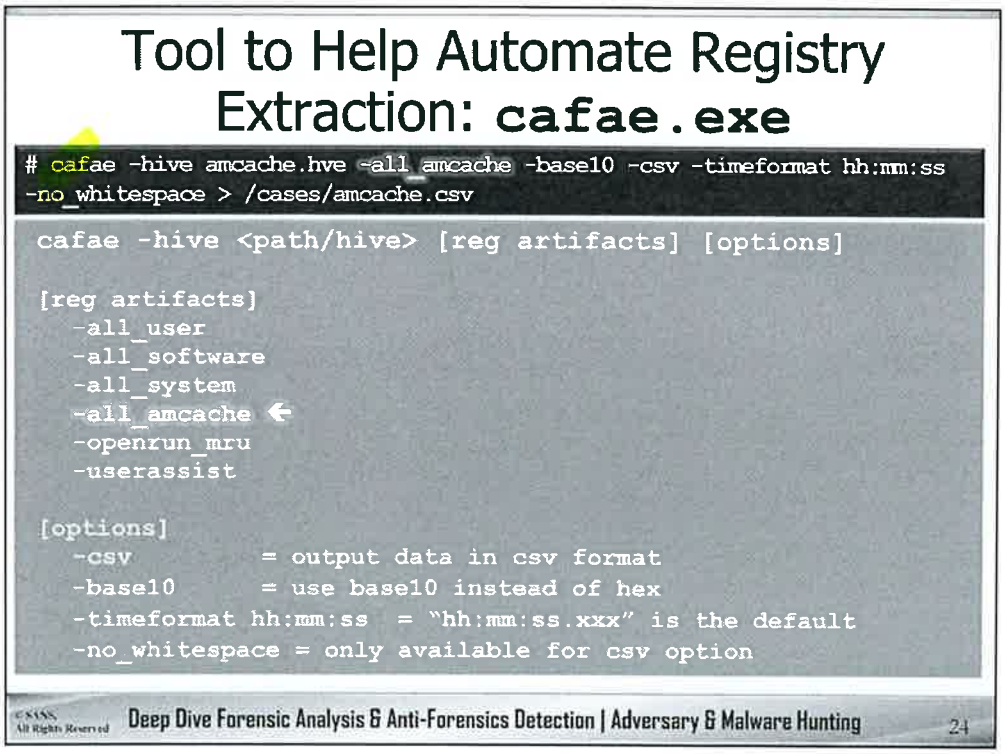


In the above graphic, there are three key items that are likely to be found in every executable entry in the Amcache.hve.

1. Executable name/path
2. First Time of execution (Last write time of key)
3. SHA1 hash of the executable – remove leading 4 zeros before submitting to virustotal for lookups.

In addition, you can parse the Volume GUIDs using and correlating the information from the SYSTEM hive's MountedDevices key.


AccessData Registry Viewer
Executable Name
First Time of Execution
SHA1
Key Properties
Last Written Time
C:\Users\Donald\Downloads\ninja-setup-2.4.3.exe

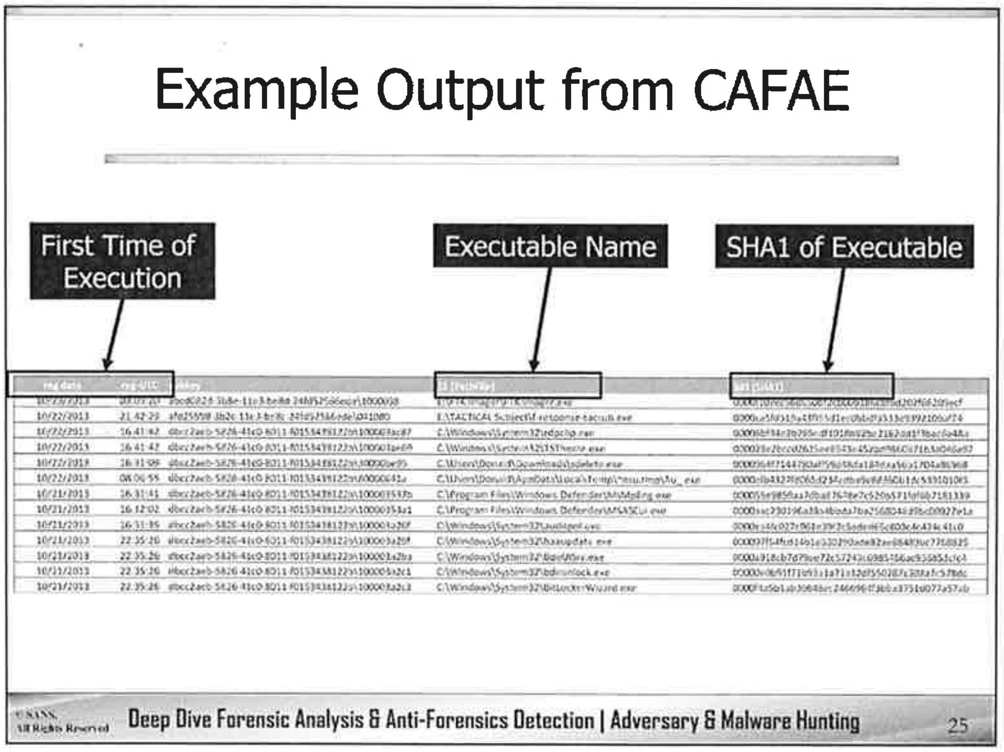


'cafae' is a registry parser written by TZWORKS LLC. cafae, which is short for Computer Account Forensic Artifact Extractor, is a Windows registry parser that targets a few specific registry keys that help identify user activity as it pertains to files and program execution. Chosen are a handful of registry entries that are specific to an account's registry hive(s). This includes both a user's ntuser.dat hive and the usrclass.dat hive for Vista and later. Collectively, these two registry hives contain artifacts useful in piecing together some sort of file/program activity that occurred on a specific account.

cafae consists of the same parsing engine that is in yaru but packaged into a console application. Consequently, the reports that are generated will look similar to those of yaru's, but will have more output options, such as a couple of CSV variants, the ability to change the date format, ability to set the time precision up to the native Windows internal precision (100 ns), and others.

Other useful aspects of cafae include the following:

- Can parse hives from a live system (same as yaru).
- Is ubiquitous across WinXP through Win8 (meaning it figures what version of the hive it is working on and then automatically adjusts which registry keys should be used).
- In some cases, it can parse deeper into the metadata and pull out additional artifacts than current registry parsers available.
- Non-Windows versions are available for those that choose to process Windows artifacts on a non-Windows operating system.
- The architecture is extremely extensible to include additional registry subkeys.
- New capabilities include the ability to parse the Windows 8+ Amcache.hve file.



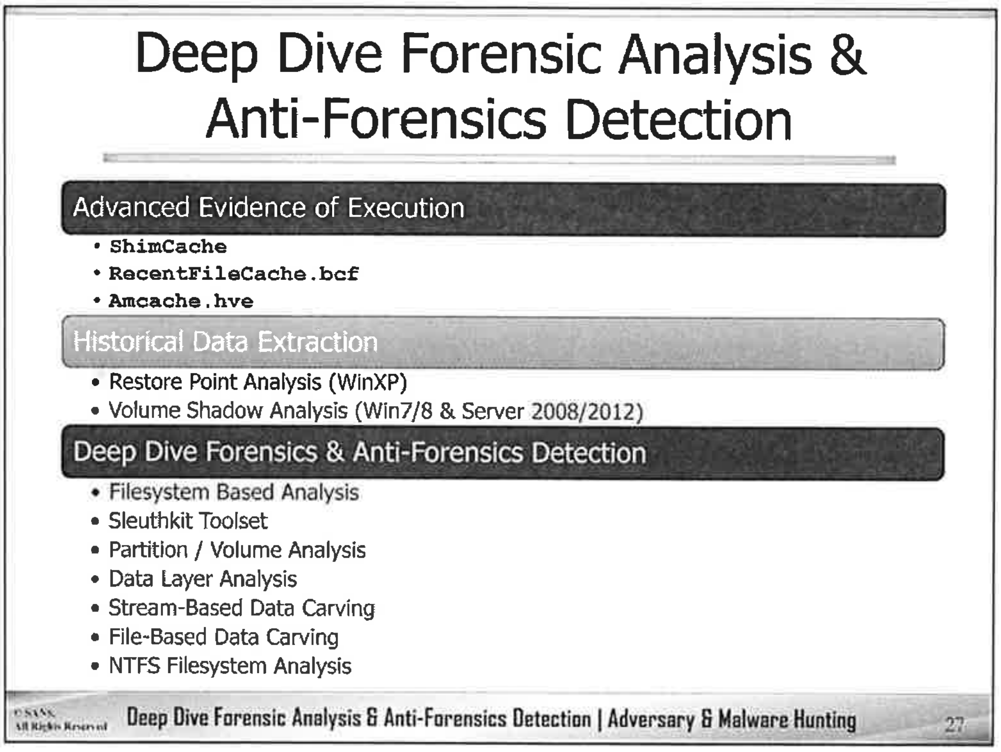
This page intentionally left blank.

First Time of Execution

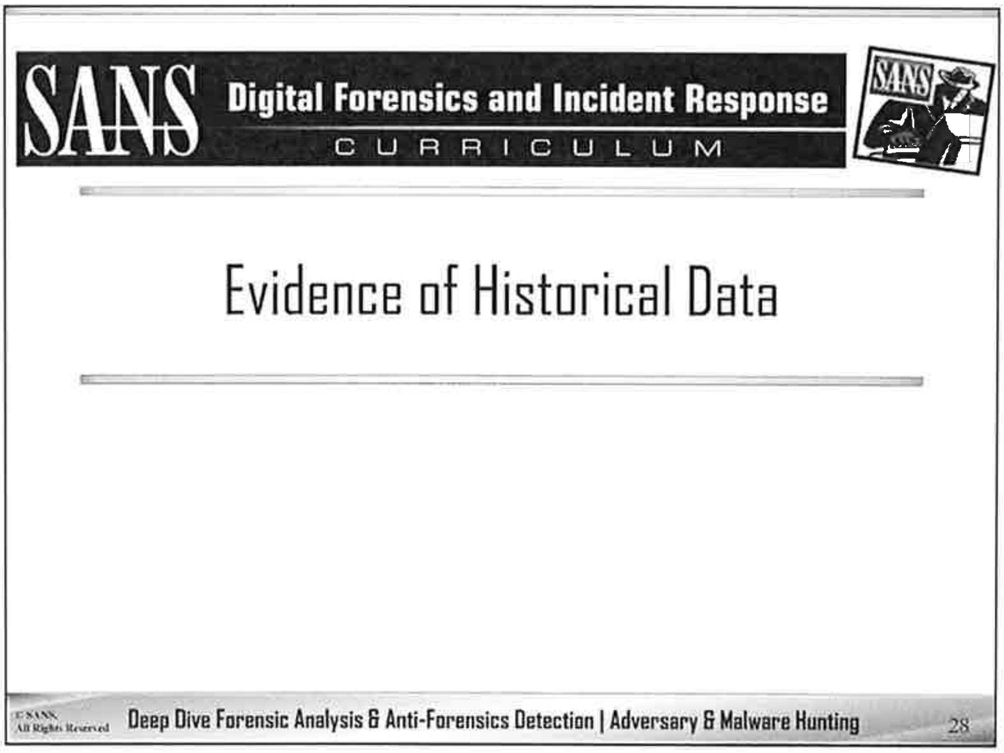
Executable Name

SHA1 of Executable

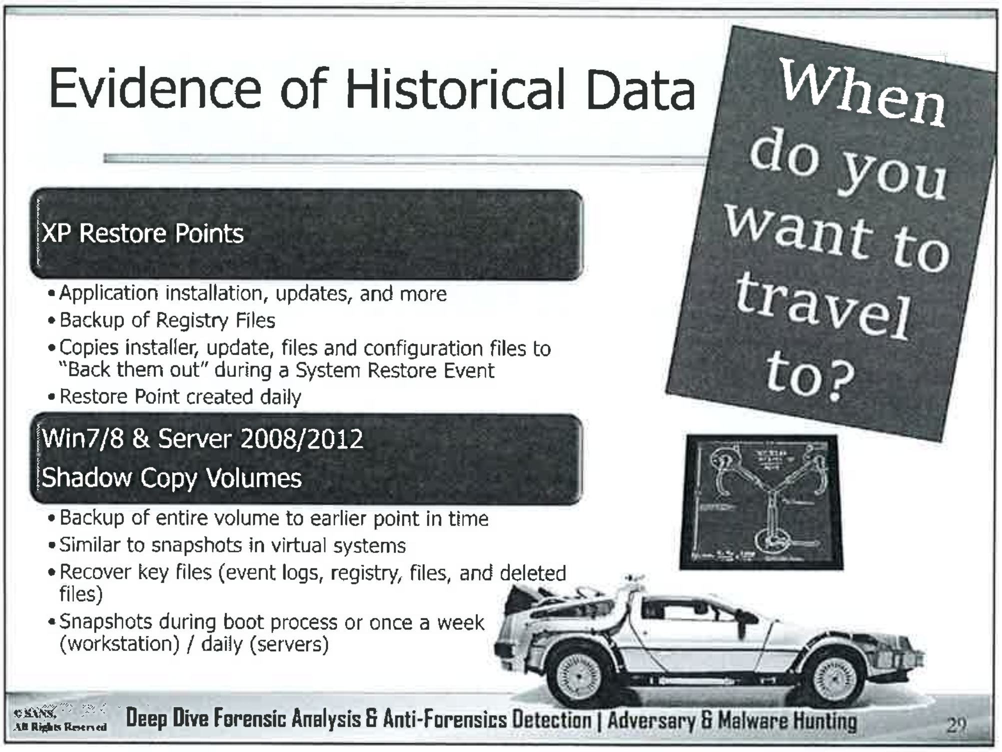
reg date	reg-UTC	subkey	15 [PathFile]	101 [SHA1]
10/23/2013	03:09:20	abcd082d-3b8e-11e3-be8d-24fd52566ede\1000038	E:\FTK Imager\FTK Imager.exe	0000f107ec56d650bf2cb00b186cbfbd202f66209ecf
10/22/2013	21:42:29	afd25598-3b2c-11e3-be8c-24fd52566ede\041080	E:\TACTICAL Subject\f-response-tacsub.exe	0000ca5fd519a43ff95d1ec0bbdf3533e9392109af74
10/22/2013	16:41:42	dbcc2aeb-5826-41c0-8011-f0153438122b\100003ac87	C:\Windows\System32\rdpclip.exe	00006bf34e2b765edf1018b82be2162dd1f3bac6a48a
10/22/2013	16:41:42	dbcc2aeb-5826-41c0-8011-f0153438122b\100003ae69	C:\Windows\System32\TSTheme.exe	000028e2bcc02625ee8543e452bd9860871b3a046a97
10/22/2013	16:31:09	dbcc2aeb-5826-41c0-8011-f0153438122b\30000be95	C:\Users\Donald\Downloads\sdelete.exe	0000964f7144780aff59d48da184daa56b1704a86968
10/22/2013	08:06:55	dbcc2aeb-5826-41c0-8011-f0153438122b\80000641a	C:\Users\Donald\AppData\Local\Temp\~nsu.tmp\Au_.exe	0000efb4327f8065d234edba9e88360b1dc5391010f5
10/21/2013	16:31:41	dbcc2aeb-5826-41c0-8011-f0153438122b\100003537b	C:\Program Files\Windows Defender\MsMpEng.exe	000055e9859aa7dba87678e7c529b571fdf6b7181339
10/21/2013	16:12:02	dbcc2aeb-5826-41c0-8011-f0153438122b\10000353a1	C:\Program Files\Windows Defender\MSASCui.exe	0000aac230196a28a4bbda7ba2568048d9bc00927e1a
10/21/2013	16:31:39	dbcc2aeb-5826-41c0-8011-f0153438122b\100003a26f	C:\Windows\System32\auditpol.exe	0000ea4fc027c961e39f2c5eded65c803c4c424c41c0
10/21/2013	22:35:26	dbcc2aeb-5826-41c0-8011-f0153438122b\100003a29f	C:\Windows\System32\baaupdate.exe	000097f54fcd14b1e630290ade82ae88489bc7768825
10/21/2013	22:35:26	dbcc2aeb-5826-41c0-8011-f0153438122b\100003a2ba	C:\Windows\System32\BdeUISrv.exe	0000a918cb7d79ee72c57243c6985466ac956853cfc4
10/21/2013	22:35:26	dbcc2aeb-5826-41c0-8011-f0153438122b\100003a2c1	C:\Windows\System32\bdeunlock.exe	00000e0b91f71b93a1a71a32df550287c308a7c578dc
10/21/2013	22:35:26	dbcc2aeb-5826-41c0-8011-f0153438122b\100003a2c3	C:\Windows\System32\BitLockerWizard.exe	0000f4a5b1ab30648ec2466964f3bba3751d077a57ab



This page intentionally left blank.



This page intentionally left blank.



In many situations, attackers will do a decent job of removing or eliminating their trace artifacts from a system. Use of file wipers and data cleaners are often seen by some of the more advanced groups to hide their more sensitive capabilities. For example, if an adversary uses a local system privilege escalation tool, they might consider wiping it so responders will not discover it. In addition, adversaries often wipe the archive files (.rar) from the system they exfiltrated them from. This would prevent a responder from finding them and discovering what might have been extracted if they can extract the data from the encrypted .rar files.

These files might not be permanently lost. With the existence of XP Restore Points and its more evolved cousin, Volume Shadow Copies, it is possible to recover some of the cleared data by examining evidence of historical artifacts from earlier snapshots of the system – using System Restore.

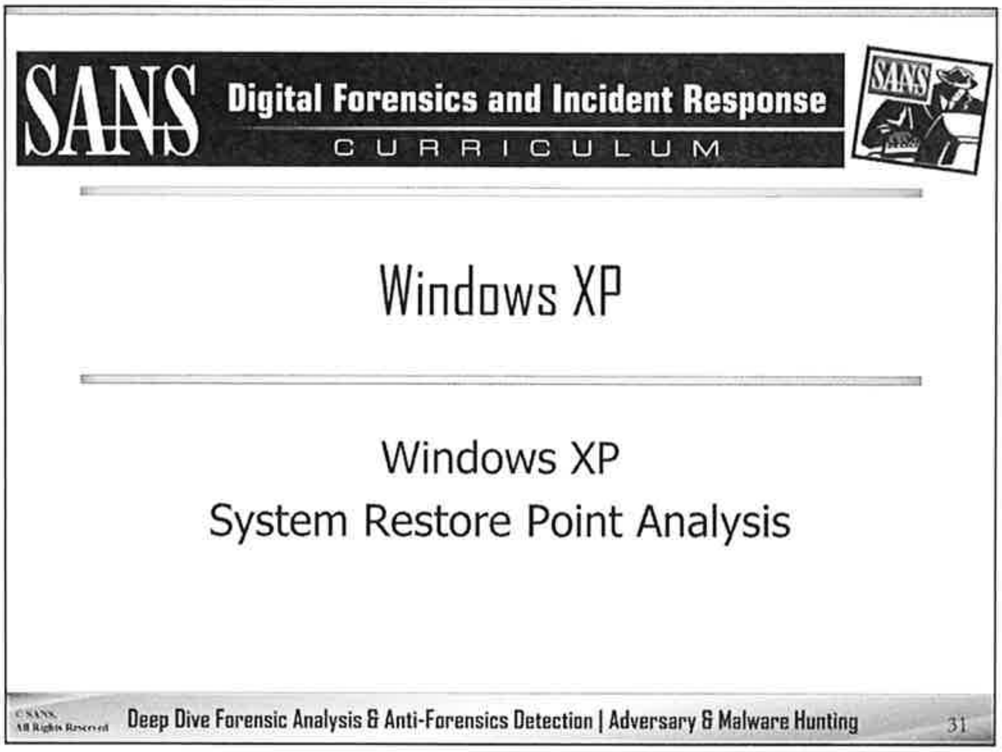
Windows XP Restore points captures application installation, updates, unsigned driver installation, and backups of registry files for each day. Likely it will also include any unsigned drivers (possible malware) loaded by an adversary.

Windows 7/8 and Server 2008/2012 uses Volume Shadow Copies to backup the entire volume. Usually this is useful if you want to recover **any** file. Key files that we can now access going back weeks include hiberfil.sys (older memory images), registry hive files, event logs, and any file that has been possibly deleted and even wiped.

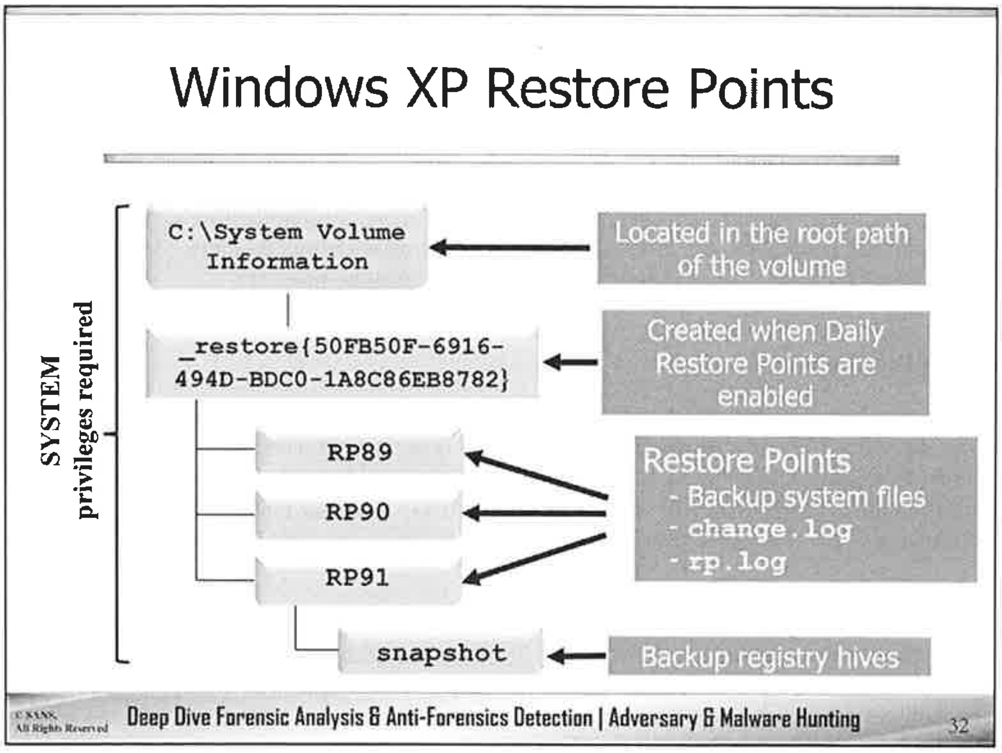
System Restore creates a restore point before the following events:

- Application installation – system restore will create a backup before the installation begins
- Unsigned Drivers – system restore will create a backup before the unsigned driver is loaded. This is particularly useful in finding evidence of malware and rootkits.

- Updates – during a system update system restore creates a restore point before the update begins
- System restore. For example, if a user accidentally chooses the wrong restore point, the user can undo the restore operation by choosing a restore point before the system restore took place. The user can then choose the correct restore point.
- System Boot – Win7/8 - volume shadow copy only



VISTA/Win7/Server 2008 Volume Shadow Copy Acquisition/Analysis.



Restore Points are generated every 24 hours a system is on. They are generally created so a user can restore his system to a previous date in case something has gone wrong. XP restore points will make up 12% of the hard drive of the system and will generally be created daily usually at night. XP restore points will also be created when a system change occurs such as a program installation, service pack update, or minor updates to the overall system.

Event-triggered restore points

System Restore automatically creates a restore point before the following events:

Application installation (provided the application utilizes an installer that is System Restore compliant). If the application install causes system problems, the user can restore the system to a state before the installation of the application.

AutoUpdate installation. AutoUpdate provides an easy way for users to download critical Windows updates. After the update is downloaded, the user can install the update on the system. If the user chooses to install the update, System Restore creates a restore point before the installation of the update begins.

System restore. For example, if a user accidentally chooses the wrong restore point, the user can undo the restore operation by choosing a restore point before the system restore took place. The user can then choose the correct restore point.

Scheduled restore points

System Restore can be configured to create restore points at regular intervals. Users can also manually create and name a restore point at any time from within the System Restore user interface. These restore points are saved and compressed, and these choices are available to the user through the System Restore user interface. System Restore in Windows 7 creates a scheduled restore point only if no other restore points have been created in the last 7 days. System Restore in Windows Vista creates a checkpoint every 24 hours if no other restore points were created that day. System Restore in Windows XP creates a checkpoint every 24 hours of absolute time. [1]

Restore Point Directory

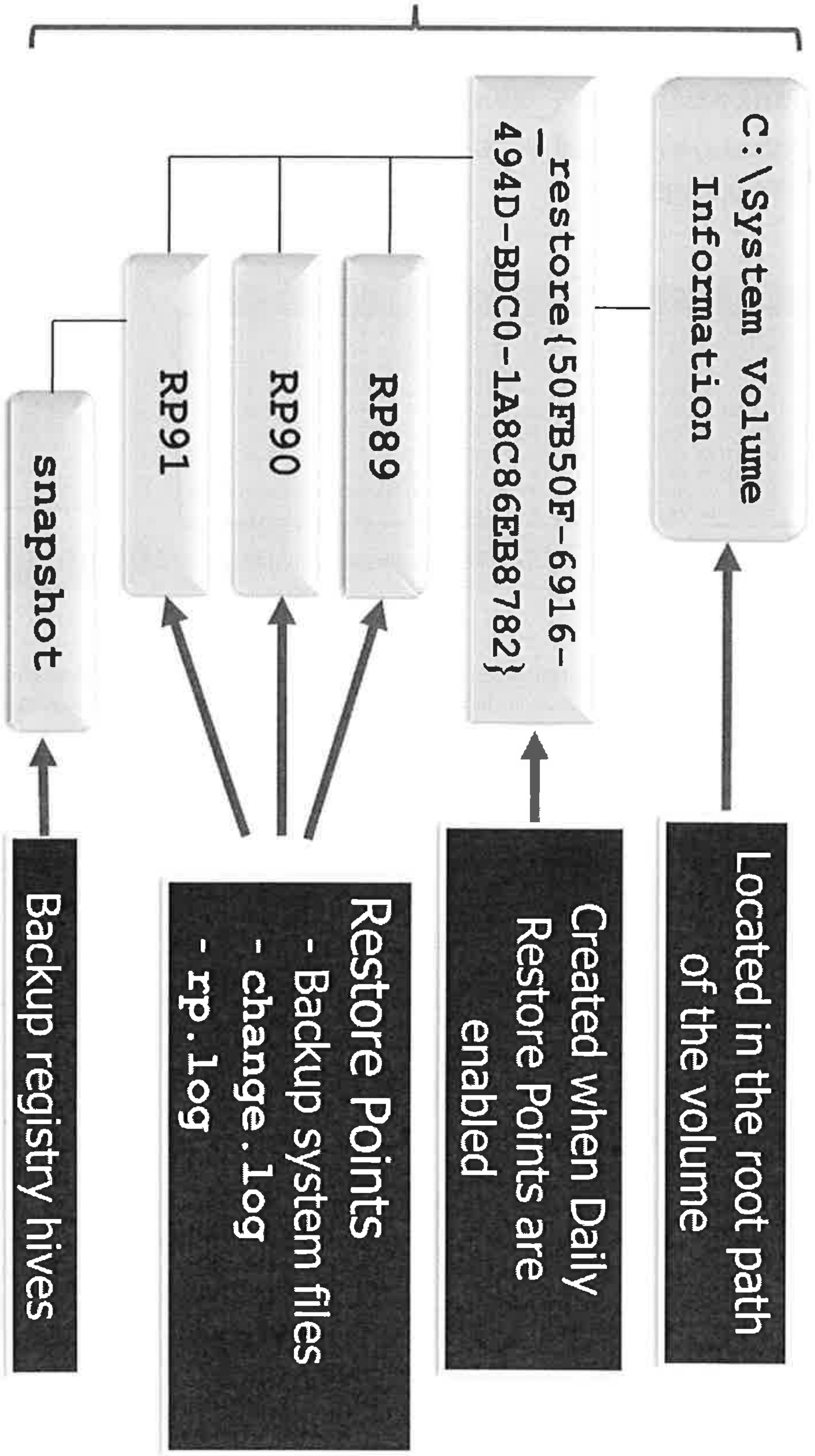
The directory where the restore points are created is in the C:\System Volume Information. This directory is only accessible by SYSTEM level privileges. Under the core C:\System Volume Information directory you will see the _restore{GUID} directory that will contain subdirectories consisting of the numbered restore points created in sequential order (e.g. RP4, RP5, RP6).

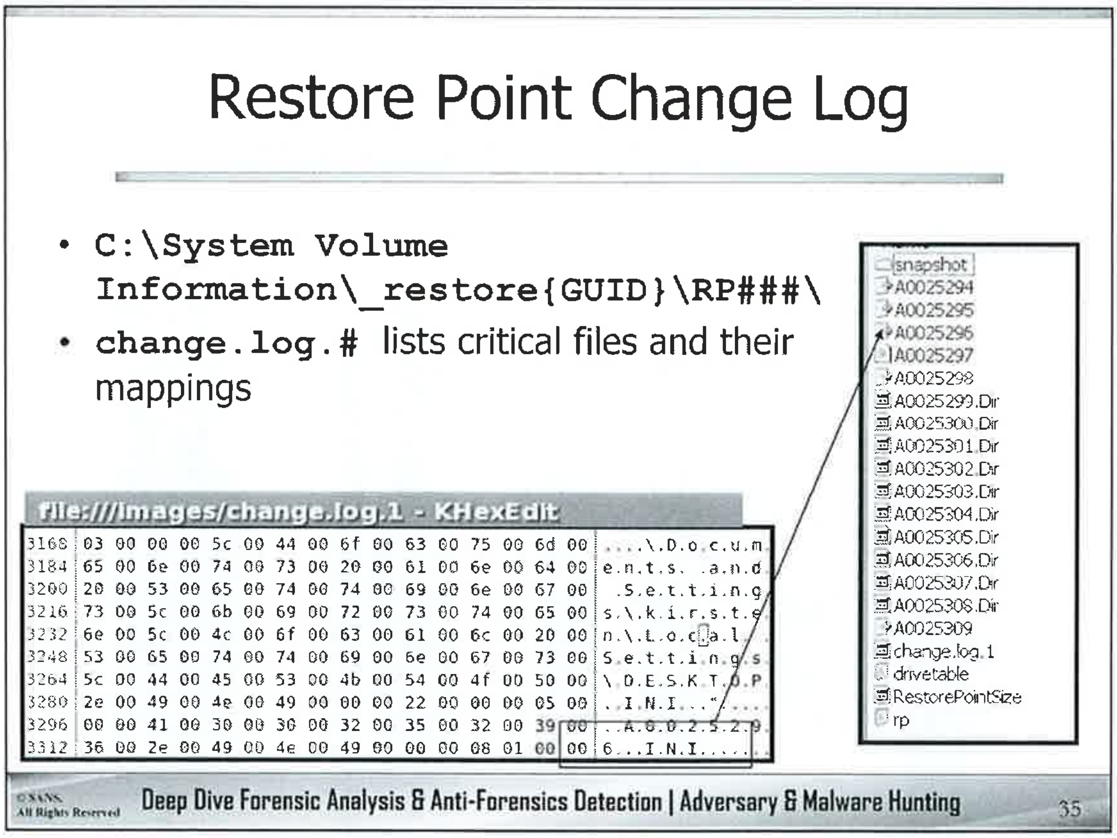
Each restore point directory will contain backup system files, and multiple log files. Files other than registry files are stored in root of folder "RP##" and renamed. They appear as A#######.ext . The "ext" represents the original extension, which remains unchanged. The change.log file will map the backup system files back to their original location. The rp.log file will detail why the restore point was created and contain the restore point creation time. The snapshot subdirectory will contain backup copies of the system hives and each of the user related hives (NTUSER.DAT, USRCLASS.DAT) for every user on the system.

It is common on many systems to possibly find restore points going back months. The historical data that is found here can be extremely useful in your cases to be able to tell what happened on a specific day as long as a restore point still exists for that day.

[1] http://msdn.microsoft.com/en-us/library/windows/desktop/aa378910%28v=vs.85%29.aspx

SYSTEM
privileges required





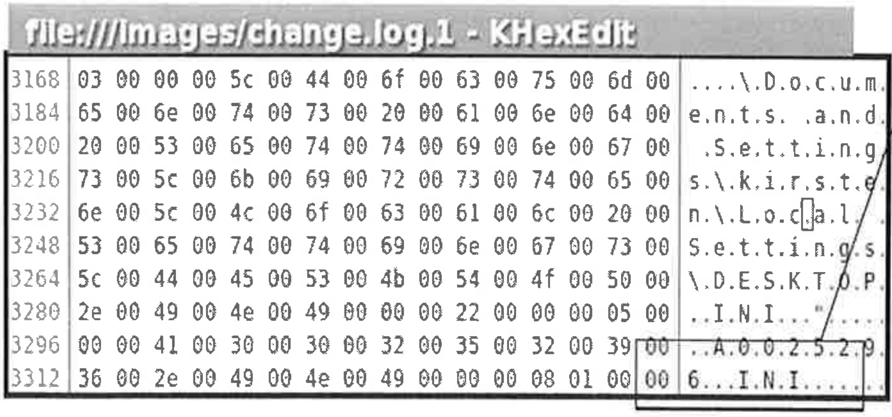
Each restore point number will also contain one or more change logs (change.log.#) that will map files stored in the snapshot back to their original locations in the file system. The files in the snapshot begin with the letter "A" and are stored in the following manner. A#######.aaa. The extension (aaa) will always remain the same as the original. In the example above we are examining the change.log.l of a snapshot where we see the file of the user's desktop.ini is stored at file A0025296.INI located in the restore point.

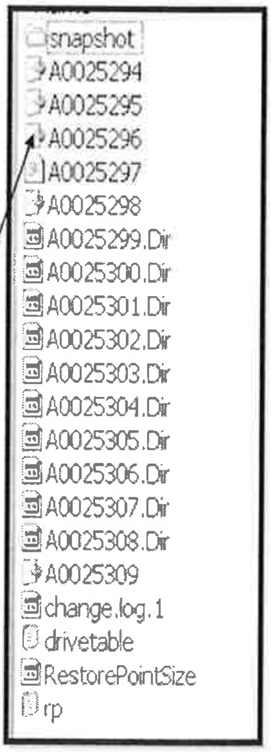
Mandiant has created a restore point analyzer that will examine the change.log and examine its contents automatically. You can easily install and use this application which is found on the DVD under the Windows forensic tools\restore point analyzer directory.

Each system restore point time is stored in the rp.log found in the RP### directory. The last 8 bytes of the rp.log will be the time the restore point was created stored in 64 bit Windows time. This time is easily converted using a tool that will decode bytes into actual times. The program is called DCode Date written by Craig Wilson (www.digital-detective.co.uk). The program is found under \windows forensic tools\hex time decoder. In this example, we examine the output from rp.log in khexedit and found the last 8 bytes were 2669b88170aec501. We plugged those bytes into the DCode Date.

Alternatively, the creation time of the rp.log file should match the embedded time inside the file itself. You really don't need to decode the internal bytes unless you absolutely want to be sure.

- `C:\System Volume Information\_restore{GUID}\RP###\`
- `change.log.#` lists critical files and their mappings





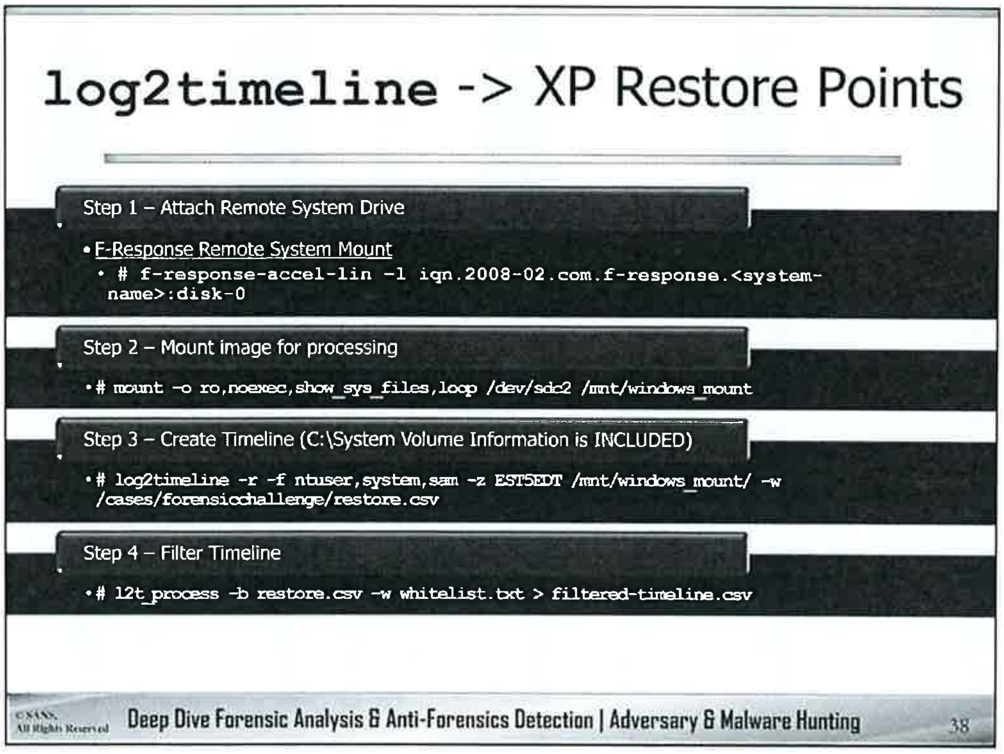
Registry of the Past?

- Restore Points and Registry
  - Collection of data points which are generated when a system restore is created every 24 hours
  - Located under a hidden folder in the volume which is easily accessed when the image is mounted for analysis

```
C:\System Volume Information\_restore{GUID}\RP###\snapshot

      SAM             _REGISTRY_MACHINE_SAM
      SECURITY        _REGISTRY_MACHINE_SECURITY
      SOFTWARE        _REGISTRY_MACHINE_SOFTWARE
      SYSTEM          _REGISTRY_MACHINE_SYSTEM
      NTUSER          _REGISTRY_MACHINE_NTUSER_SID
```


System Restore is probably one of the best places for a forensic analyst to spend a great amount of time on a Windows machine. System Restore came with XP, and ME. Windows 2003 does not have restore points. A system generally will create a restore point for every 24 hour period and it will be kept for 90 days dependent on available disk space. Granted, the user can change these values, but this is what is generally found as the default.

Restore Points will be found under the System Volume Information folder. This folder is not accessible by users or even Administrators making it extremely difficult to manipulate the values that are contained inside of this folder. Note: System account is only known means to examine this directory on a live machine.



In this example, we are using just log2timeline to sort a Windows XP restore point only looking for "Evidence of Execution" only. This is used to show how you can use log2timeline to provide a targeted timeline of only a piece of the drive image instead of the entire system itself. In this example, we could see historically the last execution time for many executables for each day a restore point was created.

Step 1 – Attach Local or Remote System Drive

```
# ewfmount system-name.E01 /mnt/ewf
```

OR

F-Response Remote System Mount

```
# f-response-accel-lin -l iqn.2008-02.com.f-response.<system-name>:disk-0
```

Step 2 – Mount image for processing

```
# mount -o ro, noexec,show_sys_files,loop /mnt/ewf/ewf1 /mnt/windows_mount
```

Step 3 – Create Timeline (C:\System Volume Information is INCLUDED)

```
# log2timeline -r -f ntuser,system,sam -z EST5EDT /mnt/windows_mount/ -w
/cases/forensicchallenge/restore.csv
```

Step 4 – Filter Timeline

```
# l2t_process -b restore.csv -w whitelist.txt > filtered-timeline.csv
```

Case Study 2: XP Restore Points

Step 1 – Find Partition Starting Sector

- # mmls image.dd - # mmls image.dd - calculate offset ##### (sector *512)

Step 2 – Mount image for processing

- # mount -o ro, noexec,show_sys_files,loop,offset=##### image.dd /mnt/windows_mount

Step 3 – Create Restore Point Timeline

- # log2timeline -r -f ntuser -z EST5EDT /mnt/windows_mount/System\ Volume\ Information/ -w /cases/forensicchallenge/restore.csv

Step 4 - Edit Keywords

- Add to keyword.txt (add userassist, runmru, LastVistedMRU, etc.)

Step 5 – Filter Timeline

- # l2t_process -b restore.csv -k keyword.txt > filtered-timeline.csv

Output from Case Study: XP Restore Points

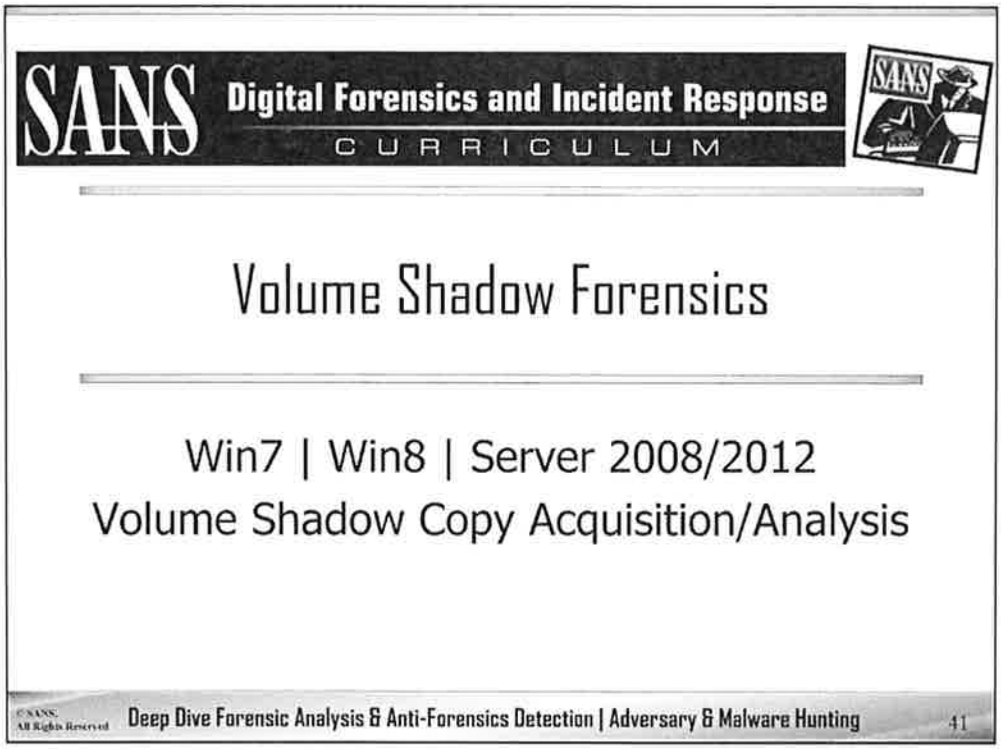
- Track Internet Explorer last execution time over time
- You can identify a specific restore point by looking at the file path that the data was culled from

date	time	sourcetype	type	desc
7/6/2008	3:49:28	UserAssist key	Time of Launch	UEME_RUNPATH:C:/Program Files/Internet Explorer/iexplore.exe [Count: 1]
7/10/2008	4:01:35	UserAssist key	Time of Launch	UEME_RUNPATH:C:/Program Files/Internet Explorer/iexplore.exe [Count: 2]
7/14/2008	23:12:45	UserAssist key	Time of Launch	UEME_RUNPATH:C:/Program Files/Internet Explorer/iexplore.exe [Count: 4]
7/15/2008	23:15:49	UserAssist key	Time of Launch	UEME_RUNPATH:C:/Program Files/Internet Explorer/iexplore.exe [Count: 5]
7/17/2008	19:49:22	UserAssist key	Time of Launch	UEME_RUNPATH:C:/Program Files/Internet Explorer/iexplore.exe [Count: 9]
7/18/2008	1:15:43	UserAssist key	Time of Launch	UEME_RUNPATH:C:/Program Files/Internet Explorer/iexplore.exe [Count: 10]

Example Only

What if a key part of our case was determining when the last time Internet Explorer was last executed over time? It is now easily visible each time IE was last executed on a specific day using timeline analysis techniques like those I showed above. Here you can easily track the execution of a specific program across multiple days thanks to quick analysis using the restore point data (NTUSER.dat hives) and **log2timeline.**

It is particularly interesting to be able to solve a restore point examination quickly using log2timeline. However, the real key to examining the data will rest in properly filtering the data using a variety of tools. As timeline becomes more popular in the future, expect that additional new parsing tools will emerge to help out with the reading of the data. In the above case, we simply used excel and used the "filtering" option to quickly identify the artifacts that are useful in our case. Learning how to filter a timeline's data is going to be key to "surviving" the data onslaught created by collecting the data into a single place.



VISTA/Win7/Server 2008 Volume Shadow Copy Acquisition/Analysis.

Shadow Copy

- You can now roll back the entire file system, a folder, or a file to a previous restoration point with Volume Shadow Service (VSS)
- Nicknamed: Shadow Copy
- When a File or Folder is modified or deleted Shadow Copy enables a user to:
  - Revert the file to any previous version
  - Restore a previous version from backup
  - Make a copy of previous version
- Shadow Copy automatically enabled in Vista/Win7
- Shadow Copy Limitations:
  - Previous version is not stored every time a user changes a file
  - Vista defaults to 15% of drive, Win7 to 3-5%(could be larger)

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 42

Windows Volume Shadow Copy is an exciting new possibility for forensic investigators looking for an additional edge in computer forensics. Restore points similar to what is found in Windows XP, do not exist in Windows Vista/Win7/Win8. Vista/Win7/Win8 now logs changes to the entire Volume and keeps track of the specific clusters that are changed on a daily basis in the new Volume Snapshot Service or VSS. Vista/Win7 Business, Enterprise, and Ultimate have Shadow Copy enabled by default.

Volume Snapshot Service (or Shadow Copy) is the new "System Restore" for Vista/Win7 and 2008. Essentially it is like "Time Machine" for MACs. It performs a cluster by cluster diffing/backup and stores that information. In a nutshell, you can rewind a file, a directory, or even an entire volume to a previous state. Wonderful for forensics.

Shadow copy enables a user to essentially revert an entire volume, a folder, or a file back in time to a previous version. An investigator can also copy out of the Shadow Copy a previous version of the file and examine the differences. VSS only takes a snapshot once a day. It will not log continuous changes every time the user saves a file.

The VSS monitors all changes made to a VSS enabled volume. These changes are monitored in 16kb 'blocks'. If a change is made to any data inside a 16kb block the entire block is copied to a volume shadow copy file. This happens regardless of the file system settings. All volume shadow copy files are stored in the 'System Volume Information' folder on the root of the volume and are recognizable by their names. The file names for volume shadow copies look something like this:

{802c6ba4-300b-11df-a523-005056c00008}{3808876b-c176-4e48-b7ae-04046e6cc752}

You can tell that this is a volume shadow copy because the second set of braces contains the number 3808876b-c176-4e48-b7ae-04046e6cc752 which is a unique identifier specific to the volume shadow service.

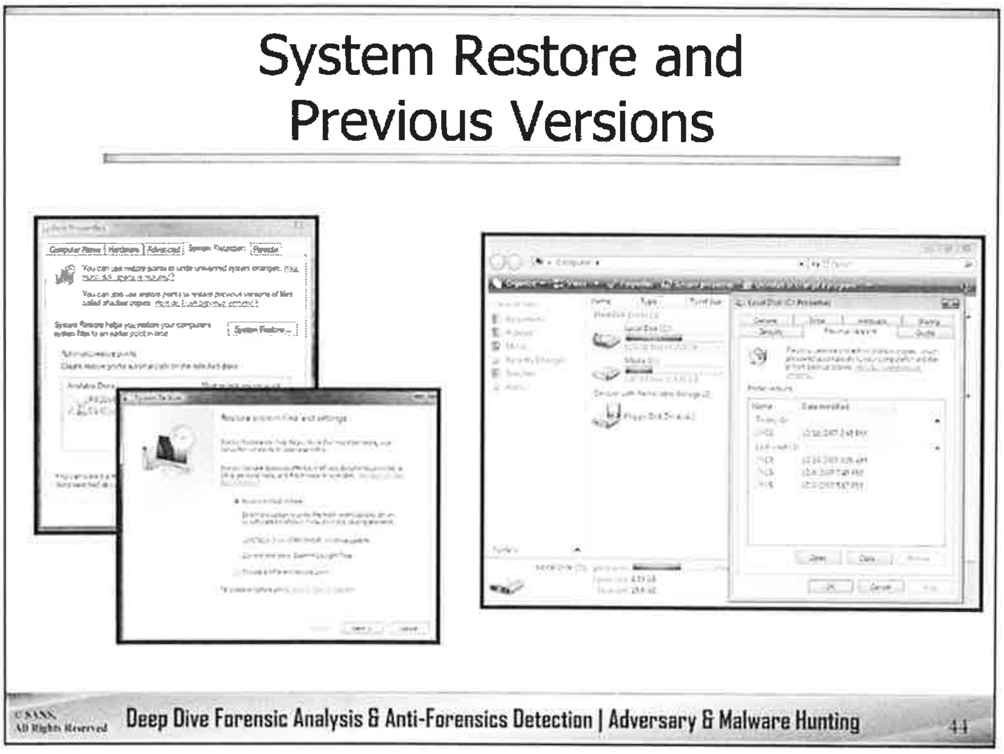
There are several ways to create a new volume shadow copy:

- System Snapshot
- Software Installation
- Manual Snapshot

On Windows Vista the system snapshot is scheduled to take place every 24 hours. On Windows 7 it is scheduled to take place every 7 days. You may notice that this is not exact. Windows Vista will not take volume snapshots exactly every 24 hours, there may be some time changes day to day. This is because the VSS will only create new shadow copies once the computer has been idle for a certain amount of time, or the computer is being turned off or rebooted, and so on.

A VSS enabled volume will always have a 'live' volume shadow copy. This file saves all of the 16kb block changes and then, once a new volume shadow copy is created, these changes are committed to the current VSC and the shadow copy is archived. From this point the contents of that volume shadow copy remains unchanged until it is deleted. [1]

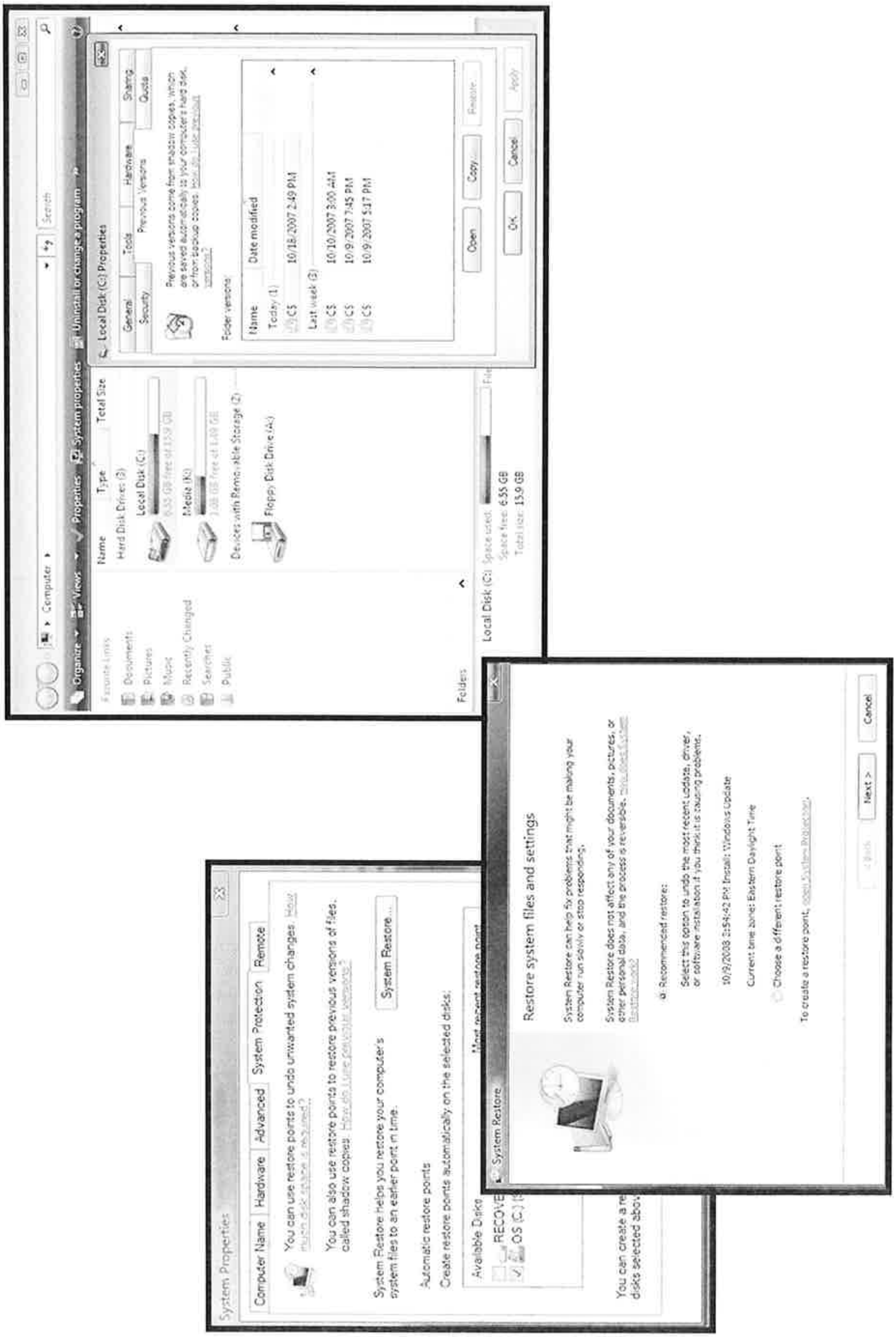
[1] Lee Whitfield – Into The Shadows Excerpt- http://www.forensic4cast.com/2010/04/into-the-shadows/

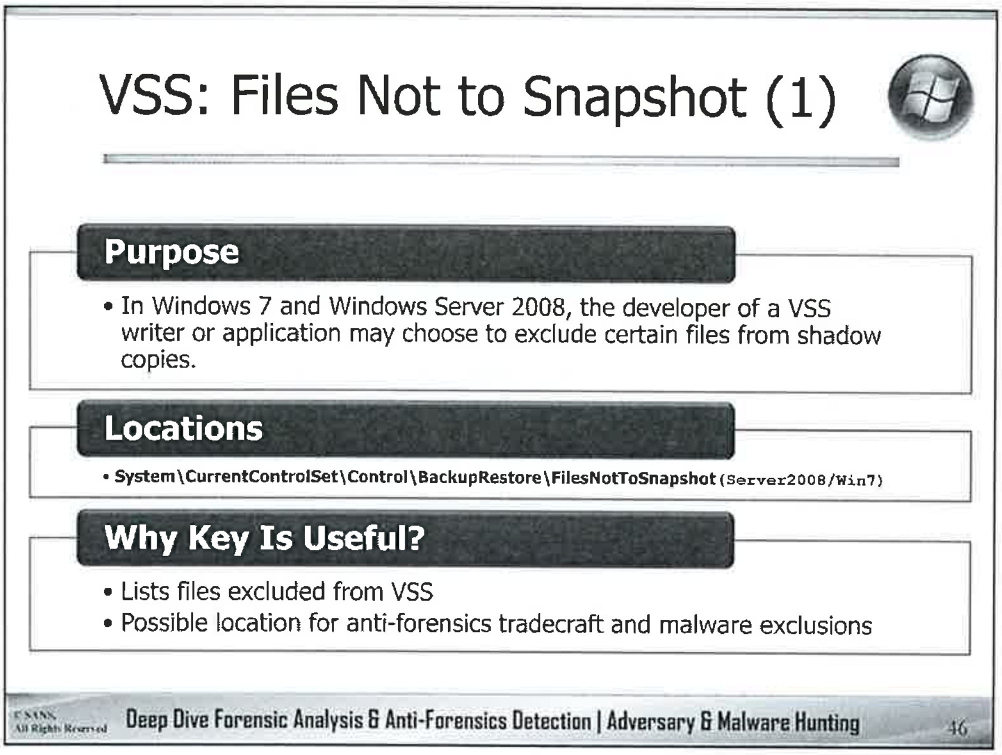


This is a snapshot of the of the restoration points created by VSS. If you notice, the entire volume of C can be backed up to earlier dates. This would mean that if a user wiped or overwrote a file yesterday with zeroes, thinking it is now gone from today's volume, it might still actually be in yesterday's snapshot. If an investigator examined the shadow volume created from yesterday's snapshot, the file is recoverable from that volume. The shadow volume that can be examined is an exact duplicate backup of the entire volume including unallocated space.

How many shadow volumes will an investigator have access to? It depends on disk size. Generally 15% of disk space is allocated for the volume shadow, however, upwards of 30% of disk space could be utilized.

VSS is utilized in two locations in VISTA. For All versions, system restore will utilize the VSS in order to back the computer up to a previous snapshot. For only business, enterprise, and ultimate versions, "Previous Versions" is enabled that will allow a user to "rewind" a file, a directory, or an entire volume.


Local Disk (C:) Properties
General
Tools
Hardware
Sharing
Security
Previous Versions
Quota
Folder versions:
Name
Date modified
Open
Copy...
Restore...
OK
Cancel
Apply
Organize
Views
Properties
System properties
Uninstall or change a program
Hard Disk Drives (3)
Local Disk (C:)
Media (R:)
Devices with Removable Storage (2)
Floppy Disk Drive (A:)
Folders
System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You can use restore points to undo unwanted system changes.
You can also use restore points to restore previous versions of files, called shadow copies.
System Restore helps you restore your computer's system files to an earlier point in time.
System Restore...
Automatic restore points
Create restore points automatically on the selected disks:
Available Disks
System Restore
Restore system files and settings
Next >
Cancel



From MSDN Article: msdn.microsoft.com/en-us/library/windows/desktop/aa819132(v=vs.85).aspx

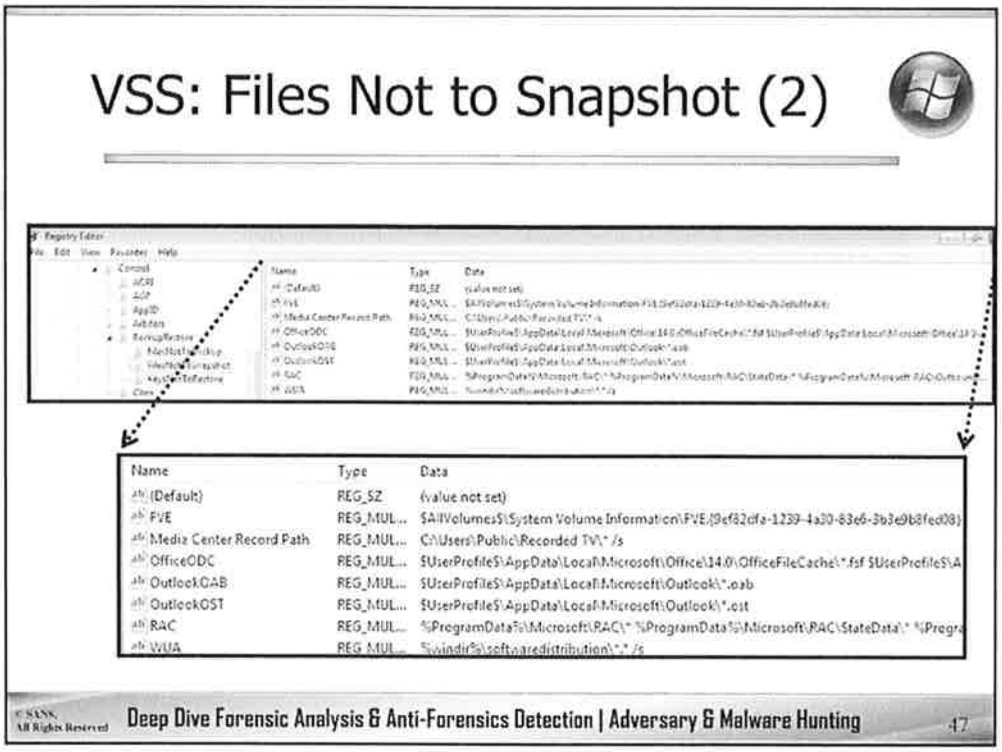
In Windows Vista and Windows Server 2008, the developer of a VSS writer or application may choose to exclude certain files from shadow copies. This could be particularly useful in anti-forensics or for ensuring exfiltrated data files are not accidentally snapshotted making recovery of what was stolen much easier.

A VSS application can delete files from a shadow copy during shadow copy creation by using the following registry key:
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\BackupRestore\FilesNotToSnapshot

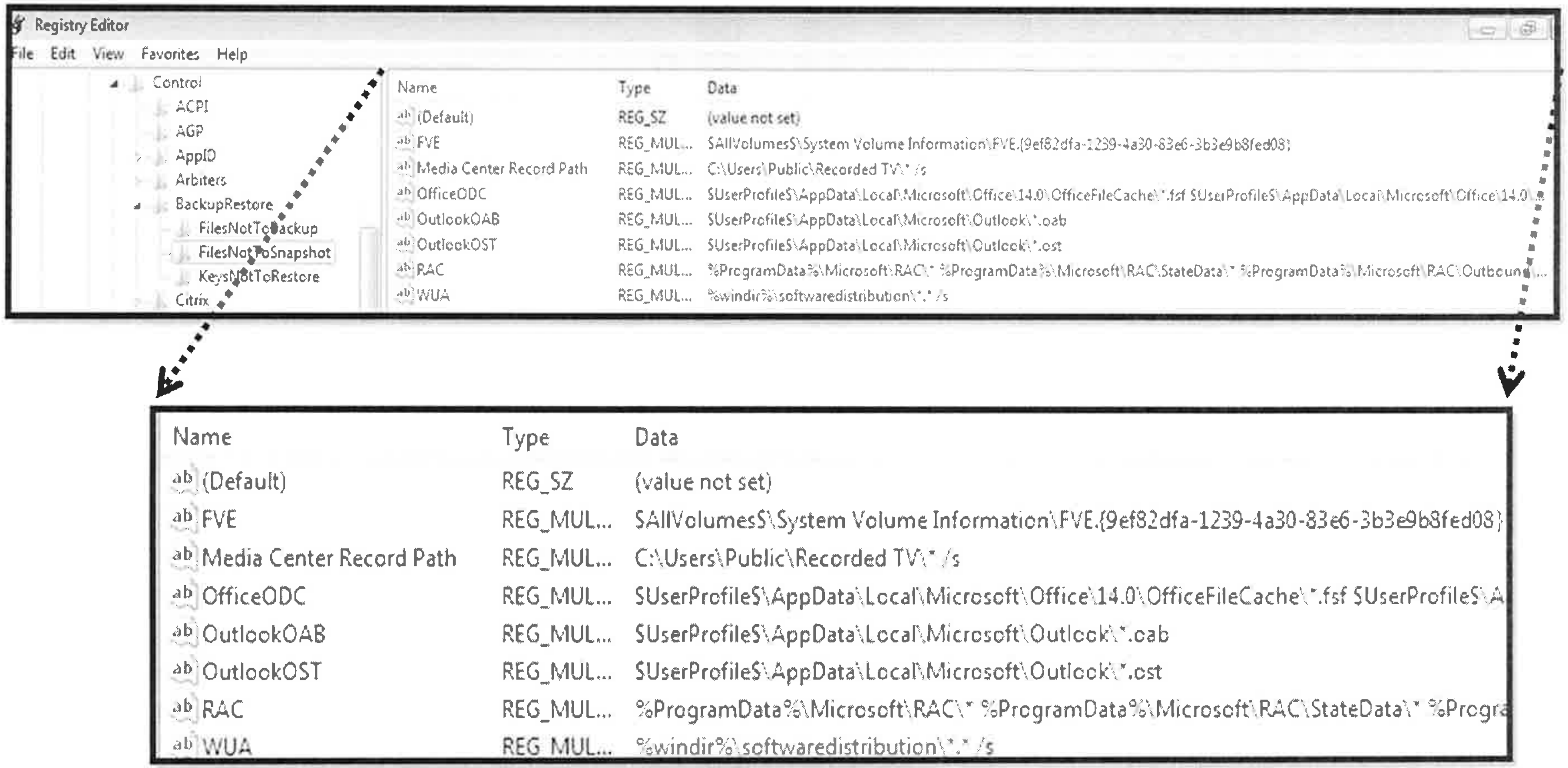
This registry key has REG_MULTI_SZ values for each application whose files can be excluded. The files are specified by fully qualified paths, which can contain the * wildcard.

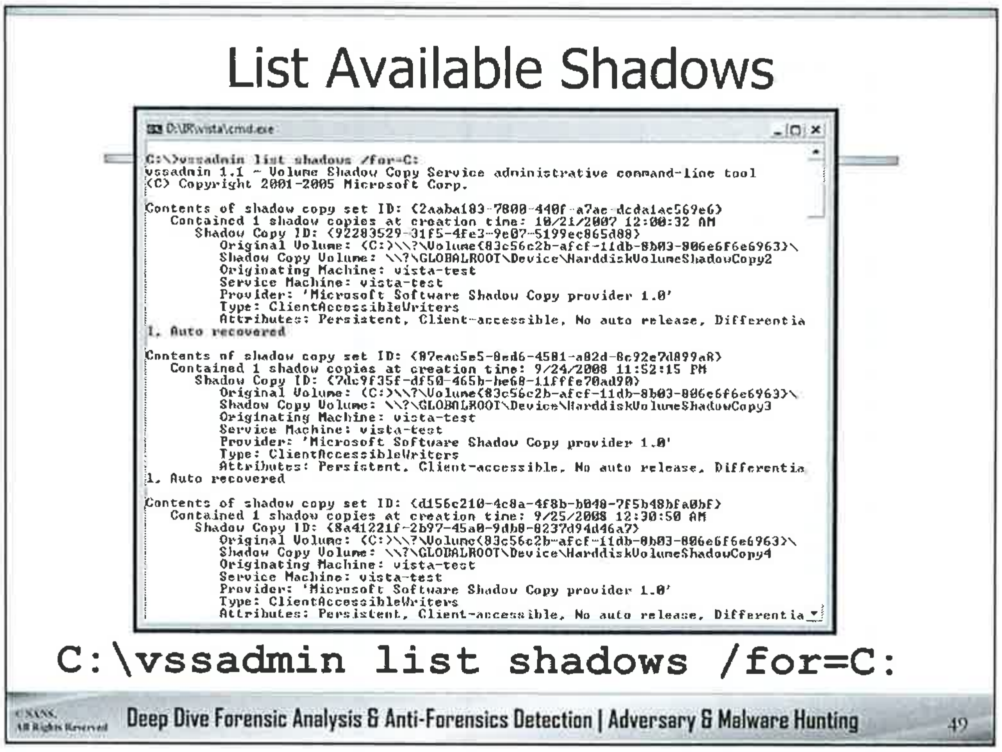
In all cases, the entry is ignored if there are no files that match the path string.

After a file is added to the appropriate registry key value, it is deleted from the shadow copy during creation by the shadow copy optimization writer on a best-effort basis.



This is a picture of the **System\CurrentControlSet\Control\BackupRestore\FilesNotToSnapshot** registry location.


Registry Editor
File Edit View Favorites Help
Control
ACPI
AGP
AppID
Arbiters
BackupRestore
FilesNotToBackup
FilesNotToSnapshot
KeysNotToRestore
Citrix
Name Type Data
(Default) REG_SZ (value not set)
FVE REG_MUL... $AllVolumes$\System Volume Information\FVE.{9ef82dfa-1239-4a30-83e6-3b3e9b8fed08}
Media Center Record Path REG_MUL... C:\Users\Public\Recorded TV* /s
OfficeODC REG_MUL... $UserProfile$\AppData\Local\Microsoft\Office\14.0\OfficeFileCache*.fsf $UserProfile$\AppData\Local\Microsoft\Office\14.0\...
OutlookOAB REG_MUL... $UserProfile$\AppData\Local\Microsoft\Outlook*.oab
OutlookOST REG_MUL... $UserProfile$\AppData\Local\Microsoft\Outlook*.ost
RAC REG_MUL... %ProgramData%\Microsoft\RAC* %ProgramData%\Microsoft\RAC\StateData* %ProgramData%\Microsoft\RAC\Outbound\...
WUA REG_MUL... %windir%\softwaredistribution*.* /s
Name Type Data
(Default) REG_SZ (value not set)
FVE REG_MUL... $AllVolumes$\System Volume Information\FVE.{9ef82dfa-1239-4a30-83e6-3b3e9b8fed08}
Media Center Record Path REG_MUL... C:\Users\Public\Recorded TV* /s
OfficeODC REG_MUL... $UserProfile$\AppData\Local\Microsoft\Office\14.0\OfficeFileCache*.fsf $UserProfile$\A
OutlookOAB REG_MUL... $UserProfile$\AppData\Local\Microsoft\Outlook*.oab
OutlookOST REG_MUL... $UserProfile$\AppData\Local\Microsoft\Outlook*.ost
RAC REG_MUL... %ProgramData%\Microsoft\RAC* %ProgramData%\Microsoft\RAC\StateData* %Progra
WUA REG_MUL... %windir%\softwaredistribution*.* /s



How many Volumes are stored on the system you are examining? You can obtain a list of existing shadow volumes in the Volume Shadow copy Service by executing the tool, vssadmin.

To obtain a list of the shadows execute:

```
C:\>  vssadmin list shadows /for=C:
```

This command will list the available shadows for the C-Drive volume.

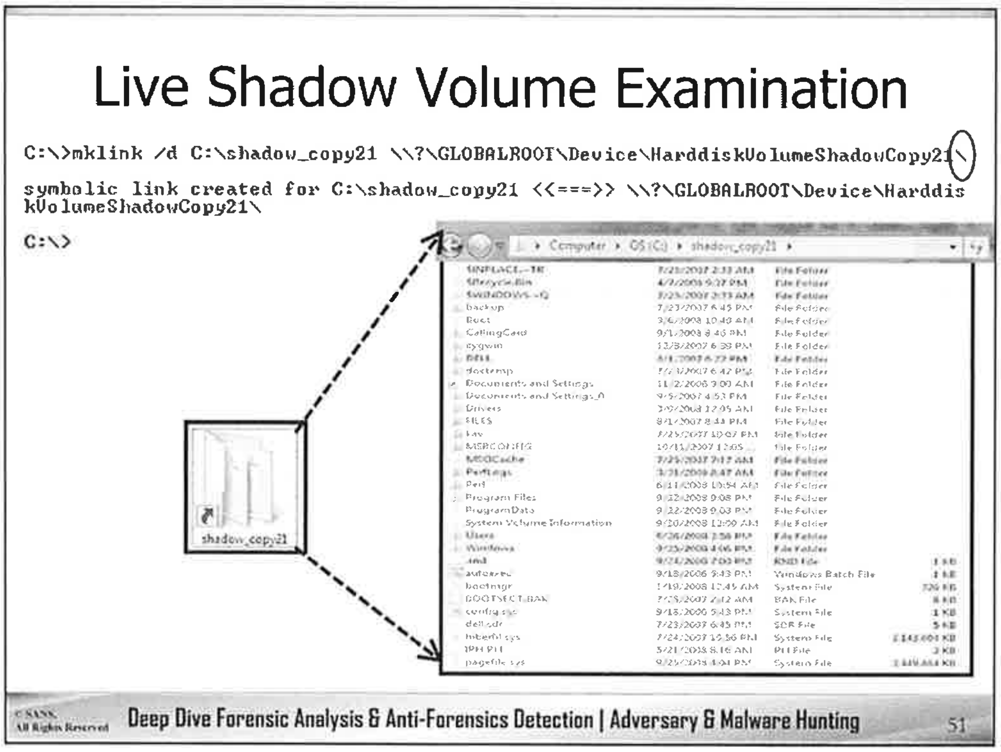
Things to notice:

1. The Shadow Copy Volume is the name of the volume that we will use to examine the contents of that specific volume. You might want to write the exact name down.
2. The originating machine would be noteworthy if you have plugged in an NTFS drive from another shadow copy enabled machine.
3. The system time of the creation time of the volume will tell you when the snapshot was created. This time is important as you know which shadow copy volume might contain your data.

From the output of vssadmin, note the total number of shadow copy volume's from the machine. In this example, it only shows 3. But there were 15 total shadow copy volumes that were listed as a result of running the "vssadmin list shadows" command. This particular machine had a 700 GB partition volume allocated for the c-drive. If you decide to image the shadow copy volumes, you could theoretically have over 16 separate 700 GB logical images created from this one machine. Each one from a different point in time.

D:\IR\vista\cmd.exe

```
C:\>vssadmin list shadows /for=C:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Contents of shadow copy set ID: {2aaba183-7800-440f-a7ae-dcda1ac569e6}
   Contained 1 shadow copies at creation time: 10/21/2007 12:00:32 AM
      Shadow Copy ID: {92283529-31f5-4fe3-9e07-5199ec865d88}
         Original Volume: (C:)\\?\Volume{83c56c2b-afcf-11db-8b03-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
         Originating Machine: vista-test
         Service Machine: vista-test
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
l, Auto recovered

Contents of shadow copy set ID: {87eac5e5-8ed6-4581-a82d-8c92e7d899a8}
   Contained 1 shadow copies at creation time: 9/24/2008 11:52:15 PM
      Shadow Copy ID: {7dc9f35f-df50-465b-be68-11fffe70ad90}
         Original Volume: (C:)\\?\Volume{83c56c2b-afcf-11db-8b03-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy3
         Originating Machine: vista-test
         Service Machine: vista-test
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
l, Auto recovered

Contents of shadow copy set ID: {d156c210-4c8a-4f8b-b048-7f5b48bfa0bf}
   Contained 1 shadow copies at creation time: 9/25/2008 12:30:50 AM
      Shadow Copy ID: {8a41221f-2b97-45a0-9db8-8237d94d46a7}
         Original Volume: (C:)\\?\Volume{83c56c2b-afcf-11db-8b03-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy4
         Originating Machine: vista-test
         Service Machine: vista-test
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
```



On a live machine it might be useful to manually browse or scan a directory that contains a shadow copy volume. It is relatively easy to do this from an administrator enabled command prompt using the tool mklink. mklink creates symbolic links, which are new to VISTA/Win7, from the command line. You can create a symbolic link from a shadow copy volume to your desktop easily by executing the following command.

From your previous output of vssadmin, select one of the "Shadow Copy Volume" names based off of the date in time you would like to examine. Then create the symbolic link using mklink pointing it at a directory followed by the device name of the shadow copy volume you wish to parse.

```
C:\>  mklink /d C:\shadow_copy21
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy21\
```

Note: It is VERY important to ensure that the trailing BACKSLASH is in the command. Without it you might not be able to access the link you created.

This capability would be incredibly useful in situations where a bad guy might have wiped or overwritten with zeros to the file. You could recover it by going back a day or two in the shadow copy volume to retrieve the file from allocated space.

```
C:\>mklink /d C:\shadow_copy21 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy21\

symbolic link created for C:\shadow_copy21 <<===>> \\?\GLOBALROOT\Device\Harddis
kVolumeShadowCopy21\

C:\>
```

$INPLACE.~TR	7/25/2007 2:33 AM	File Folder	
$Recycle.Bin	4/7/2008 9:37 PM	File Folder	
$WINDOWS.~Q	7/25/2007 2:33 AM	File Folder	
backup	7/23/2007 6:45 PM	File Folder	
Boot	3/6/2008 10:40 AM	File Folder	
CallingCard	9/1/2008 8:46 PM	File Folder	
cygwin	12/3/2007 6:39 PM	File Folder	
DELL	8/1/2007 6:22 PM	File Folder	
doctemp	7/23/2007 6:42 PM	File Folder	
Documents and Settings	11/2/2006 9:00 AM	File Folder	
Documents and Settings_0	9/5/2007 4:53 PM	File Folder	
Drivers	3/9/2008 12:05 AM	File Folder	
FILES	8/1/2007 8:44 PM	File Folder	
kav	7/25/2007 10:07 PM	File Folder	
MSBCONFIG	10/11/2007 12:05 ...	File Folder	
MSOCache	7/25/2007 7:17 AM	File Folder	
PerfLogs	3/31/2008 8:47 AM	File Folder	
Perl	6/11/2008 10:54 AM	File Folder	
Program Files	9/22/2008 9:08 PM	File Folder	
ProgramData	9/22/2008 9:08 PM	File Folder	
System Volume Information	9/30/2008 12:00 AM	File Folder	
Users	6/26/2008 3:56 PM	File Folder	
Windows	9/25/2008 4:06 PM	File Folder	
.rnd	9/24/2008 7:00 PM	RND File	1 KB
autoexec	9/18/2006 5:43 PM	Windows Batch File	1 KB
bootmgr	1/19/2008 12:45 AM	System File	326 KB
BOOTSECT.BAK	7/25/2007 2:42 AM	BAK File	8 KB
config.sys	9/18/2006 5:43 PM	System File	1 KB
dell.sdr	7/23/2007 6:45 PM	SDR File	5 KB
hiberfil.sys	7/24/2007 10:56 PM	System File	3,143,604 KB
IPH.PH	5/21/2008 8:16 AM	PH File	2 KB
pagefile.sys	9/25/2008 4:04 PM	System File	3,449,884 KB

Imaging a Shadow Volume

```
F:\fau\FAU.x86>dd if=\\.\HarddiskVolumeShadowCopy4 of=F:\snapshot4.img --localw
t
Copying \\.\HarddiskVolumeShadowCopy4 to F:\snapshot4.img
```

- Utilize Windows **`dd.exe`**
- Resulting image will be a full logical volume image from the VSS enabled machine

It is also rather simple to image the full volume of a specific shadow by utilizing the dd.exe command and the output from the vssadmin list shadows output.

Using the vssadmin list shadows output you again identify the shadow you wish to image. In this situation, you would just utilize the part of the device name that includes "HarddiskVolumeShadowcopy#" The # (number) is the number of the shadow copy volume you wish to collect a full disk image from.

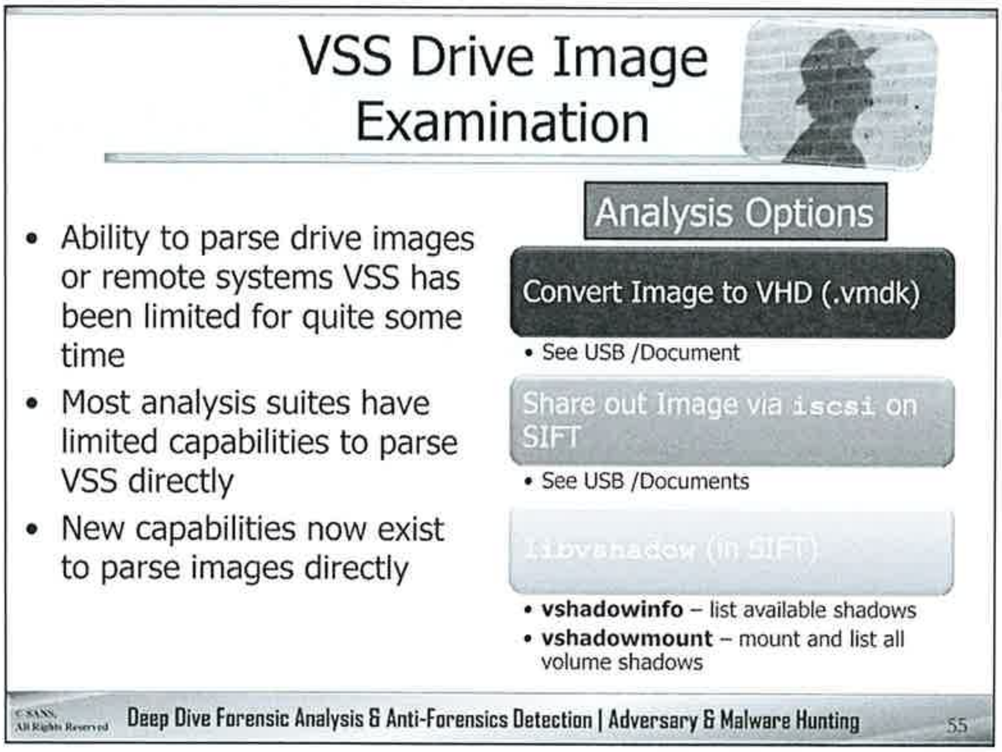
All the regular options in dd apply, but to image execute a command similar to the following on your machine that contains the volume shadow copy or the read-only original drive containing the volume shadow copy.

In this command F: is a USB drive plugged in the machine. The **`--localwrt`** option allows the dd.exe tool write to a local mounted drive.

```
dd.exe if=\\.\HarddiskVolumeShadowCopy4 of=F:\snapshot4.img  --localwrt
```

Examining Shadows Volumes in a Raw Image

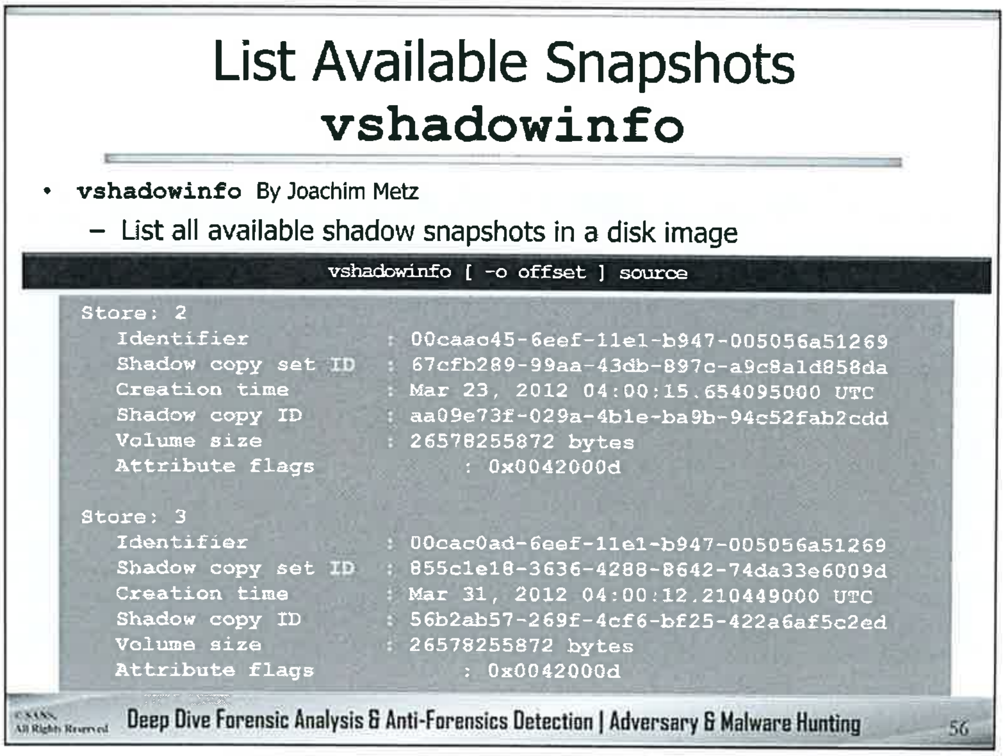
This page intentionally left blank.



Since VSS arrived in 2007, many analysts have not been able to easily parse the data contained in a volume shadow snapshot. There are many tricks to be able to get access to the volume shadows within a disk image, but until recently, very few direct processing tools have existed.

The old methodology is still to convert an image to a vmdk and load it within a virtual machine. Another method is to share out the drive via an iscsi interface to another windows machine connected to it via the iscsi initiator. Both methods are thoroughly explained in the appendices.

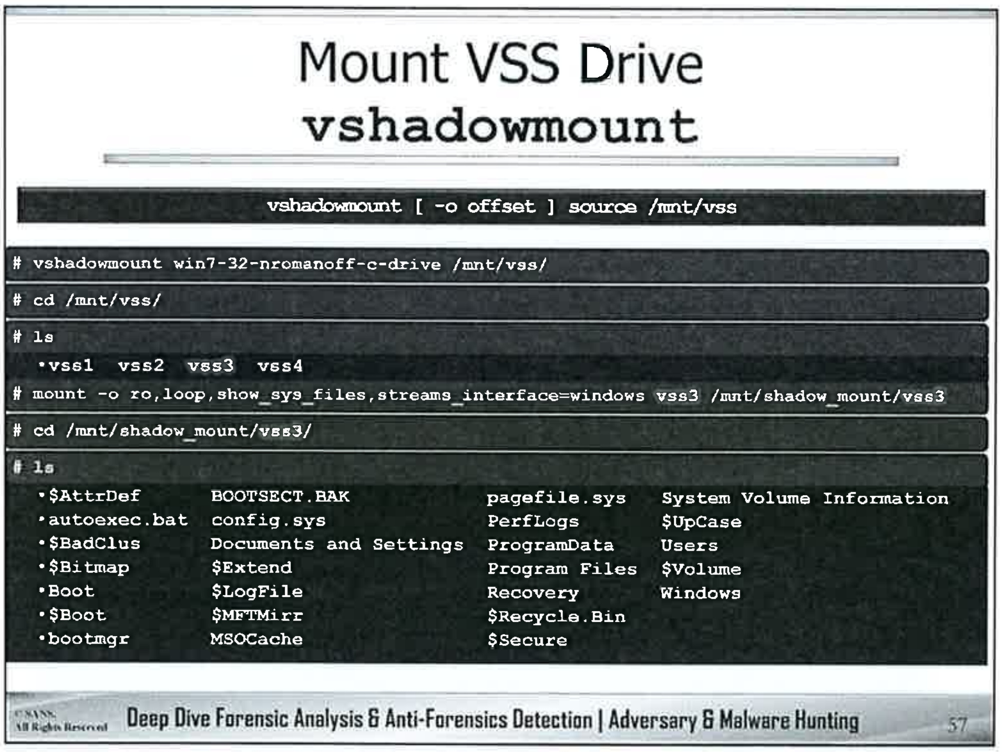
Released this year, Joachim Metz, wrote a library and several tools with the capability to parse volume shadows. This capability has been installed on the SIFT workstation. In this next section we will explore their capabilities and describe several analysis methods that will change the way you approach volume shadow analysis.



The first tool that is quite useful has the root of the capability based on the original "vssadmin list shadows" windows command. The command is vshadowino. When you point vshadowinfo at the raw disk image you can have it list out each volume snapshot and the date/time it was created on the system.

If you are attempting to parse the volume shadows of a physical disk image, the addition –o offset option should be used to point at the appropriate disk offset where the NTFS logical volume begins that contain the VSS.

Knowing the exact time and dates, it will help aid the investigator to know which snapshot might contain the data that they seek.



The second tool we will look at is the capability to mount all the VSS images so we can analyze them in the SIFT workstation. The process will first use the command vshadowmount to mount the volume shadows at the /mnt/vss directory. We can now view all the volume snapshots now available for disk mounting through using the final mount command as always.

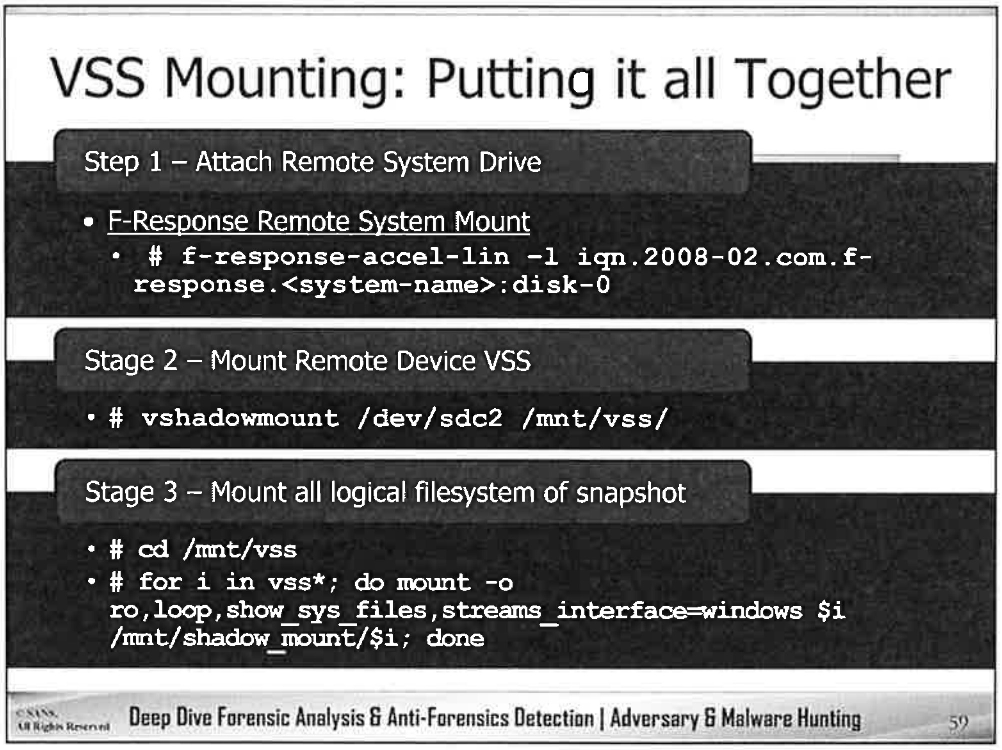
If you are attempting to mount a physical disk image, you can use the offset option (-o) to specify the disk offset where the NTFS partition begins in the volume you are attempting to mount.

`vshadowmount` By Joachim Metz

Mount a Volume Shadow Snapshot (VSS) volume

```
# vshadowmount win7-32-nromanoff-c-drive /mnt/vss/

# cd /mnt/vss/

# ls
      vss1   vss2   vss3   vss4

# mount -o ro,loop,show_sys_files,streams_interface=windows vss3
/mnt/shadow_mount/vss3
```

```
# cd /mnt/shadow_mount/vss3/

# ls
      $AttrDef      BOOTSECT.BAK          pagefile.sys    System Volume
      Information
      autoexec.bat  config.sys            PerfLogs        $UpCase
      $BadClus      Documents and Settings ProgramData    Users
      $Bitmap       $Extend               Program Files   $Volume
      Boot          $LogFile              Recovery        Windows
      $Boot         $MFTMirr              $Recycle.Bin
      bootmgr       MSOCache              $Secure
```



Stage 1 – Attach Local or Remote System Drive

```
# ewfmount system-name.E01 /mnt/ewf
```

OR

<u>F-Response Remote System Mount</u>

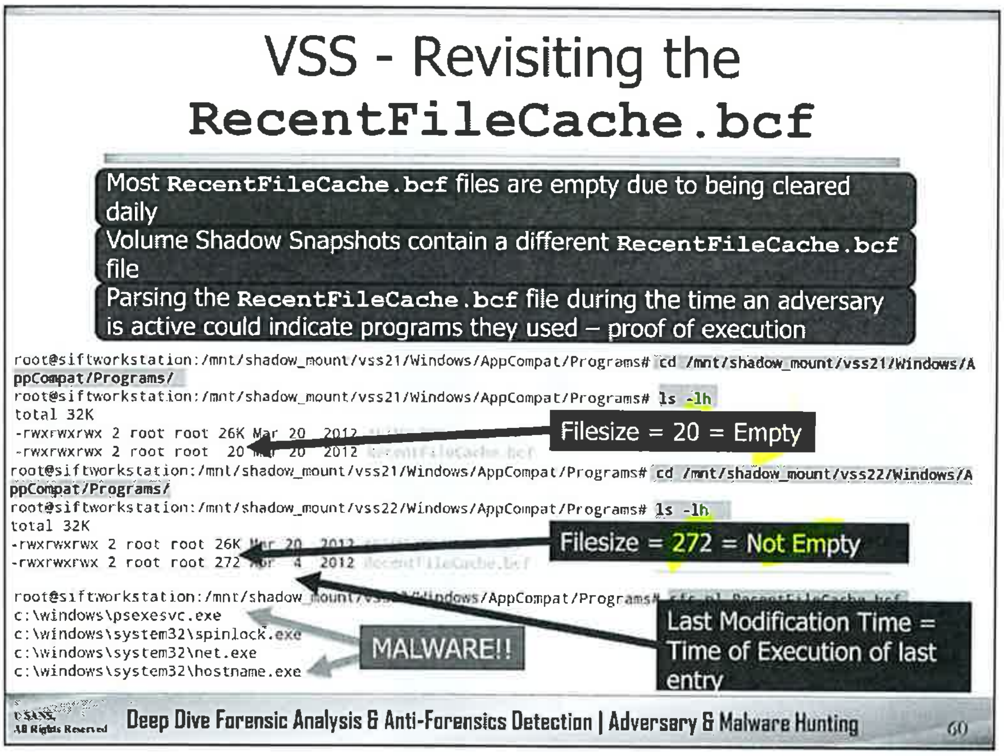
```
# f-response-accel-lin -l iqn.2008-02.com.f-response.<system-name>:disk-0
```

Stage 2 – Mount raw image VSS

```
# vshadowmount ewf1 /mnt/vss/
```

Stage 3 – Mount All Logical filesystem of snapshot

```
# cd /mnt/vss
# for i in vss*; do mount -o
ro,loop,show_sys_files,streams_interface=windows $i
/mnt/shadow_mount/$i; done
```



Remember when we mentioned that the RecentFileCache.bcf file was cleared daily? This would be frustrating as the artifact would only be good for 24 hours. Most RecentFileCache.bcf files are empty due to being cleared daily. We can tell they are empty by exploring each Volume Shadow Snapshot and seeing if the file size = 20 bytes. Volume Shadow Snapshots contain a different RecentFileCache.bcf file. Look for a RecentFileCache.bcf file that contains a larger file size across your various VSS snapshots. Then once you find one, you can parse it via RCF.pl. Parsing the RecentFileCache.bcf file during the time an adversary is active could indicate programs they used – proof of execution. The last modification time of the RecentFileCache.bcf file could possibly indicate when one or more of the executables were run. It is unknown at this time the order in which the executable files have been added to the registry hive.

```
root@siftworkstation:/mnt/shadow_mount/vss21/Windows/AppCompat/Programs# cd /mnt/shadow_mount/vss21/Windows/AppCompat/Programs/
root@siftworkstation:/mnt/shadow_mount/vss21/Windows/AppCompat/Programs# ls -lh
total 32K
-rwxrwxrwx 2 root root 26K Mar 20  2012
-rwxrwxrwx 2 root root  20 Mar 20  2012 RecentFileCache.bcf
```

Filesize = 20 = Empty

```
root@siftworkstation:/mnt/shadow_mount/vss21/Windows/AppCompat/Programs# cd /mnt/shadow_mount/vss22/Windows/AppCompat/Programs/
root@siftworkstation:/mnt/shadow_mount/vss22/Windows/AppCompat/Programs# ls -lh
total 32K
-rwxrwxrwx 2 root root 26K Mar 20  2012
-rwxrwxrwx 2 root root 272 Apr  4  2012 RecentFileCache.bcf
```

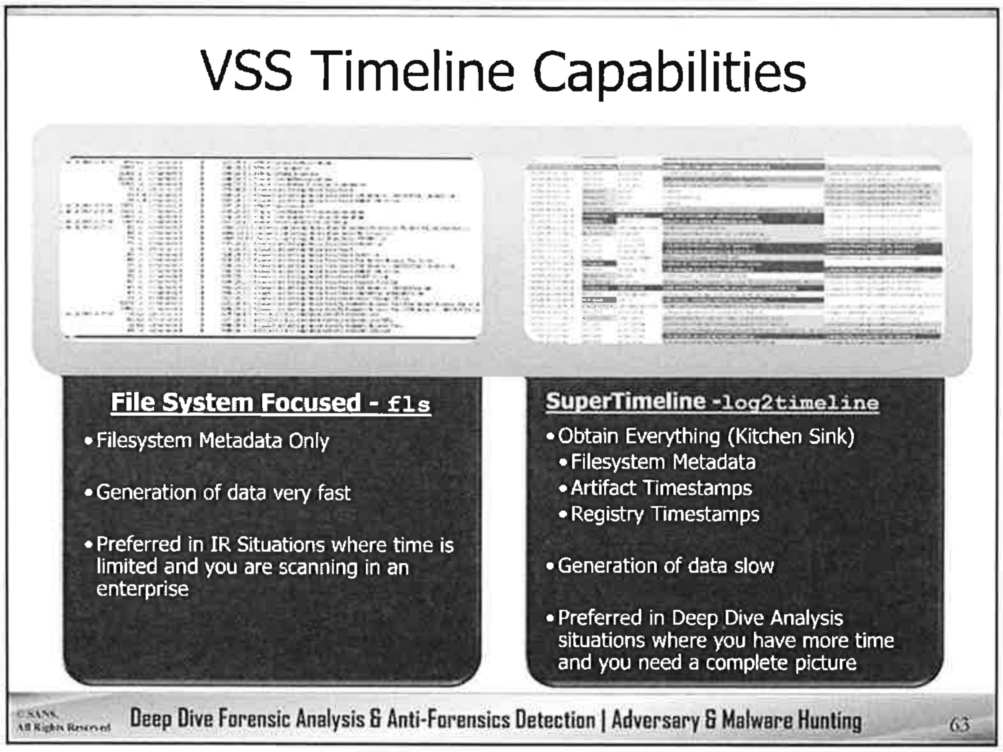
Filesize = 272 = Not Empty

```
root@siftworkstation:/mnt/shadow_mount/vss22/Windows/AppCompat/Programs# rfc.pl RecentFileCache.bcf
c:\windows\psexesvc.exe
c:\windows\system32\spinlock.exe
c:\windows\system32\net.exe
c:\windows\system32\hostname.exe
```

SANS Digital Forensics and Incident Response CURRICULUM

Shadow Timelines

This page intentionally left blank.

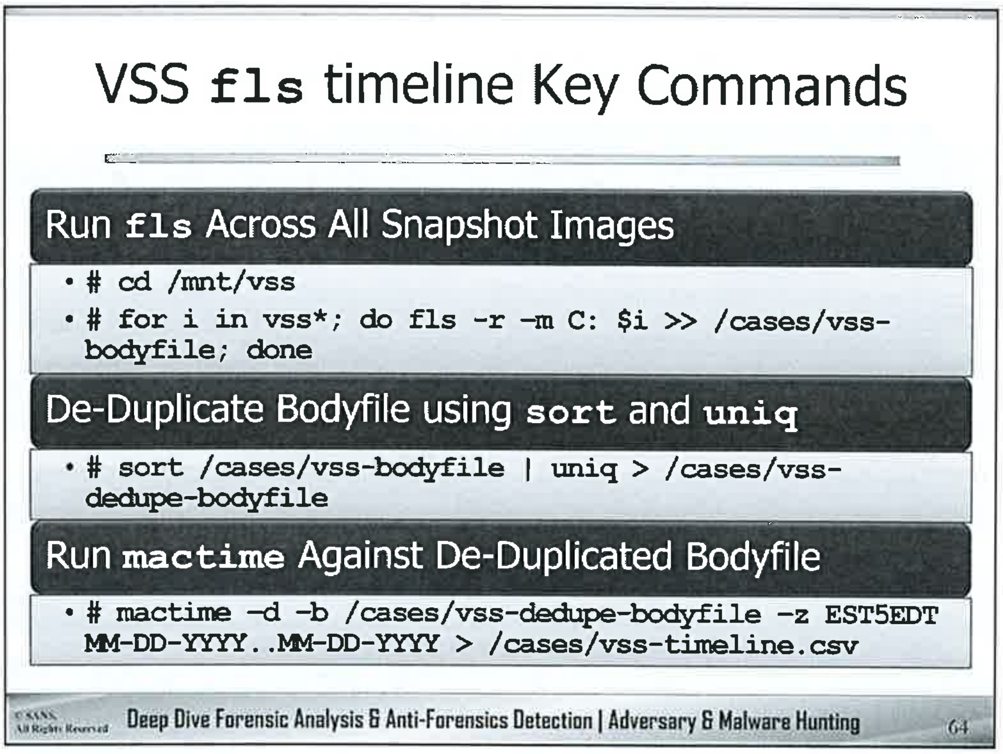


Having access to all the volume snapshots is wonderful. However, the frustration will begin when you realize you now have too much data to analyze. Timeline analysis can make short work of a simple, yet powerful approach to analyzing the data stored across all of your volume shadow snapshots.

It would be fairly time intensive for you to manually examine each snapshot as if it were a single system. Knowing this, it is more efficient to reduce the amount of data by de-duplicating the entries that are found in all the snapshots and only focus on the ones that are different or add more clarity to the examination by revealing items that may have been changed, deleted, or wiped at an earlier point in time.

There are two methods to be considered. The fls method, or filesystem, is much quicker and could be processed in less than a half hour for a single system. The super timeline, log2timeline/plaso method, could take days to complete. A single drive might take a single day. It is possible to focus your log2timeline only on the most important artifacts instead of using the overall (system based) list files. For example, use log2timeline to extract MFT, Registry, and EVTX data across all your snapshots only.

If in incident response situations where time is of the essence and you want to peer into the VSS quickly to see if anything can be used, I would recommend the fls – filesystem method. If you are performing deep-dive analysis of a system and you have a day to spare to create it, I would recommend the log2timeline method discussed.



Run `fls` Across All Snapshot Images

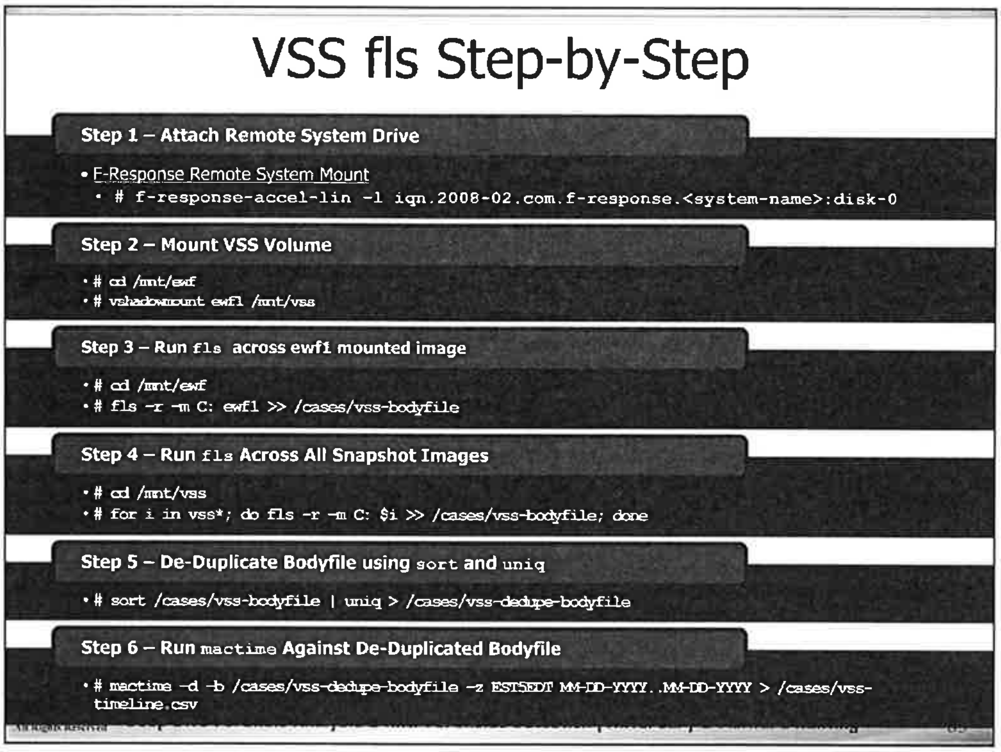
```
# cd /mnt/vss
# for i in vss*; do fls -r -m C: $i >> /cases/vss-bodyfile; done
```

De-Duplicate Bodyfile using `sort` and `uniq`

```
# sort /cases/vss-bodyfile | uniq > /cases/vss-dedupe-bodyfile
```

Run `mactime` Against De-Duplicated Bodyfile

```
# mactime -d -b /cases/vss-dedupe-bodyfile -z EST5EDT MM-DD-YYYY..MM-DD-YYYY > /cases/vss-timeline.csv
```



Overall, the filesystem method is much quicker than the log2timeline method. As a result, it is possible to use the fls method in an enterprise scan. However, it is not recommended to use this across every system, just the ones that are the most likely to be compromised. This "quick look" deep analytical capability will give you additional information especially if you have a decent pivot point to work from where you know something "Evil" probably happened at a specific time on the system.

Step 1 – Attach Local or Remote System Drive

```
# ewfmount system-name.E01 /mnt/ewf
```

OR

F-Response Remote System Mount

```
# iscsiadm -m node --targetname=iqn.2008-02.com.f-response.talon:disk-0 -login
```

Step 2 – Mount VSS Volume

```
# cd /mnt/ewf
# vshadowmount ewf1 /mnt/vss
```

Step 3 – Run `fls` across ewf1 mounted image

```
# cd /mnt/ewf
# fls -r -m C: ewf1 >> /cases/vss-bodyfile
```

Step 4 – Run `fls` Across All Snapshot Images

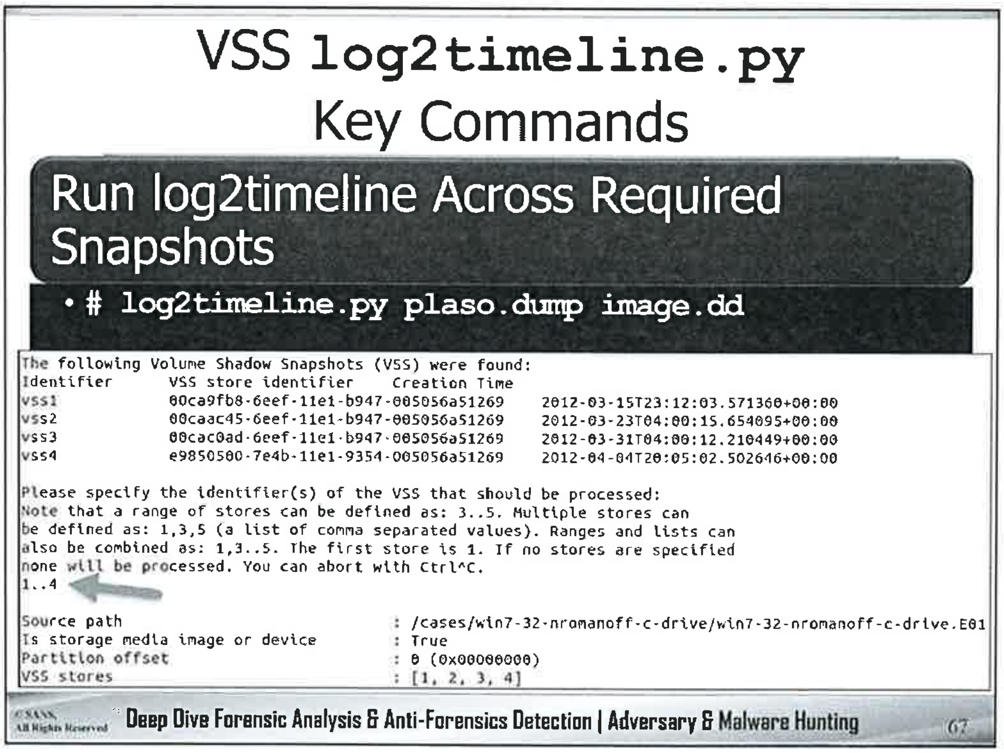
```
# cd /mnt/vss
# for i in vss*; do fls -r -m C: $i >> /cases/vss-bodyfile; done
```

Step 5 – De-Duplicate Bodyfile using `sort` and `uniq`

```
# sort /cases/vss-bodyfile | uniq > /cases/vss-dedupe-bodyfile
```

Step 6 – Run `mactime` Against De-Duplicated Bodyfile

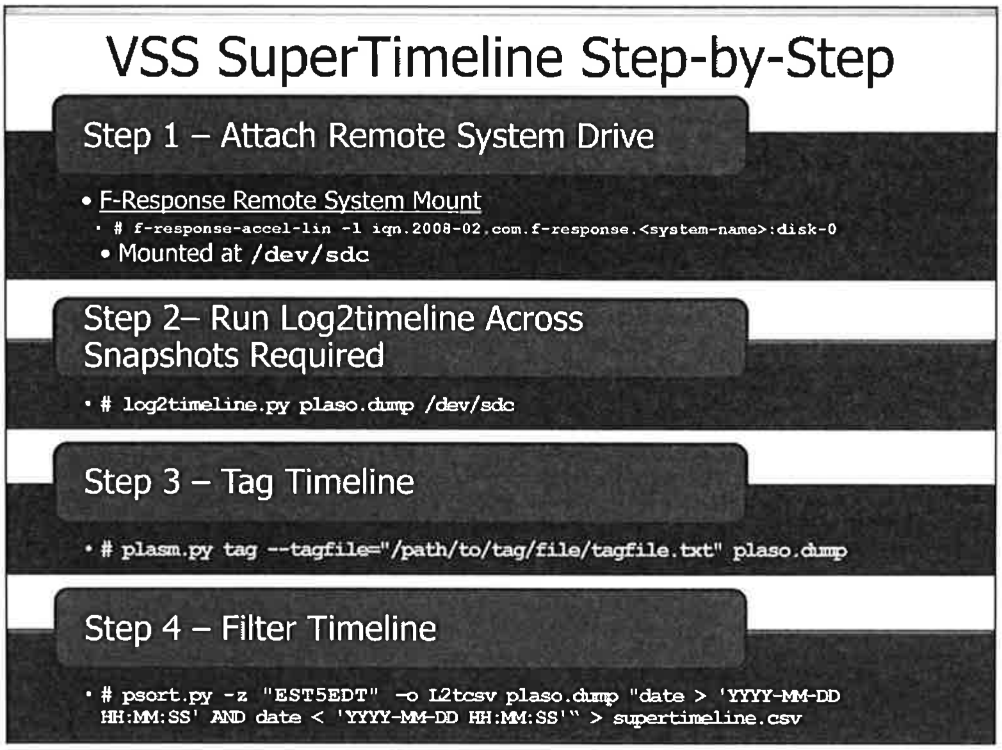
```
# mactime -d -b /cases/vss-dedupe-bodyfile -z EST5EDT MM-DD-
YYYY..MM-DD-YYYY > /cases/vss-timeline.csv
```



Run Log2timeline Across All Mounted Snapshots

```
# cd /mnt/vss
# log2timeline.py plaso.dump image.dd

The following Volume Shadow Snapshots (VSS) were found:
Identifier          VSS store identifier    Creation Time
vss1                00ca9fb8-6eef-11e1-b947-005056a51269
      2012-03-15T23:12:03.571360+00:00
vss2                00caac45-6eef-11e1-b947-005056a51269
      2012-03-23T04:00:15.654095+00:00
vss3                00cac0ad-6eef-11e1-b947-005056a51269
      2012-03-31T04:00:12.210449+00:00
vss4                e9850500-7e4b-11e1-9354-005056a51269
      2012-04-04T20:05:02.502646+00:00

Please specify the identifier(s) of the VSS that should be processed:
Note that a range of stores can be defined as: 3..5. Multiple stores
can be defined as: 1,3,5 (a list of comma separated values). Ranges
and lists can also be combined as: 1,3..5. The first store is 1. If
no stores are specified none will be processed. You can abort with
Ctrl^C.
```

```
YOU TYPE 1..4

Source path                                           : /cases/win7-
32-nromanoff-c-drive/win7-32-nromanoff-c-drive.E01
Is storage media image or device                               :
True
Partition offset                                      : 0
(0x00000000)
VSS stores                                            : [1, 2, 3, 4]
```



Please note: The following will take a considerable amount of time (possibly even a day to complete) if you use the default system list files. It is highly recommended that instead of using the log2timeline list files (win7, winxp) that you should only perform a targeted extraction and only add data as you need it. In the above example, we are only pulling timeline data for mft, ntuser, system, software, evtx, and webhistory.

Step 1 – Attach Remote System Drive

F-Response Remote System Mount

```
# f-response-accel-lin -l iqn.2008-02.com.f-response.<system-name>:disk-0
```

Mounted at `/dev/sdc`

Step 2– Run Log2timeline Across All Mounted Snapshots

```
# log2timeline.py plaso.dump /dev/sdc
```

Step 3 – Tag Timeline

```
# plasm.py tag --tagfile="/path/to/tag/file/tagfile.txt" plaso.dump
```

Step 4 – Filter Timeline

```
# psort.py -z "EST5EDT" -o L2tcsv plaso.dump "date > 'YYYY-MM-DD HH:MM:SS' AND date < 'YYYY-MM-DD HH:MM:SS'" > supertimeline.csv
```

Step 4 – Mount VSS Volume

```
# cd /mnt/ewf
# vshadowmount ewf1 /mnt/vss
```

Step 5 – Mount All VSS Snapshots

`# cd /mnt/vss` (Note: You must be in this directory)

```
# for i in vss*; do mount -o
ro,loop,show_sys_files,streams_interface=windows $i
/mnt/shadow_mount/$i; done
```

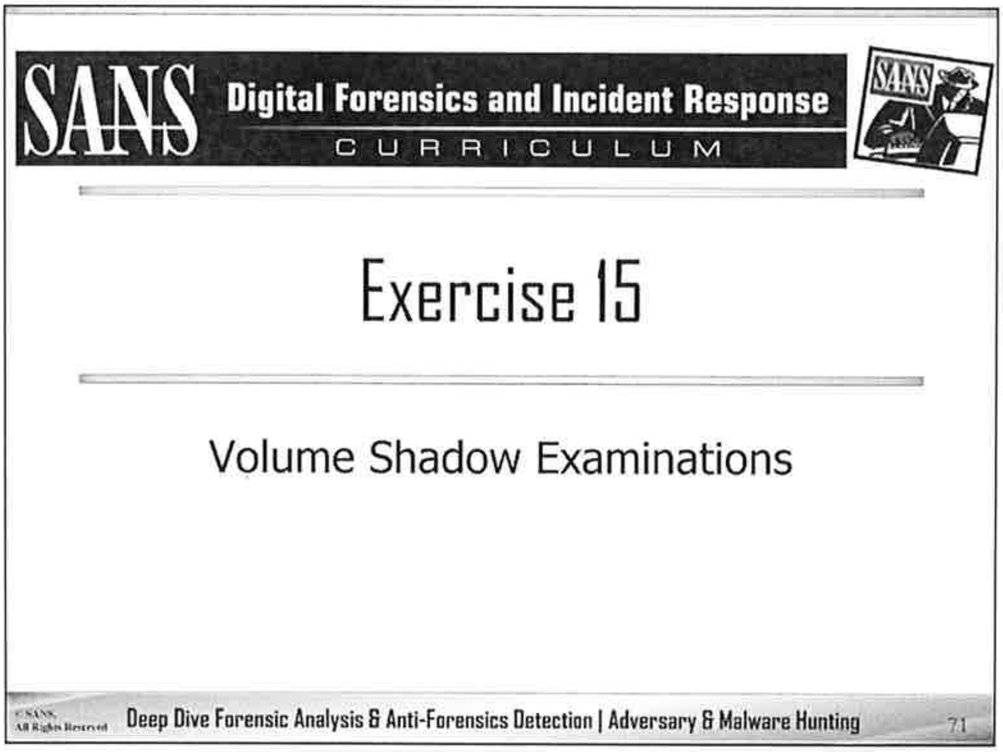
Step 6 – Run Log2timeline Across All Mounted Snapshots

`# cd /mnt/vss` (Note: You must be in this directory)

```
# for i in vss*; do log2timeline -r -p -f
mft,ntuser,system,software,evtx,webhist -z EST5EDT -w /cases/vss-
supertimeline.txt /mnt/shadow_mount/$i; done
```

Step 7 – Dedupe and Filter Timeline

```
# l2t_process -b timeline.csv -k keywords.txt MM-DD-YYYY..MM-DD-YYYY
```



This page intentionally left blank.

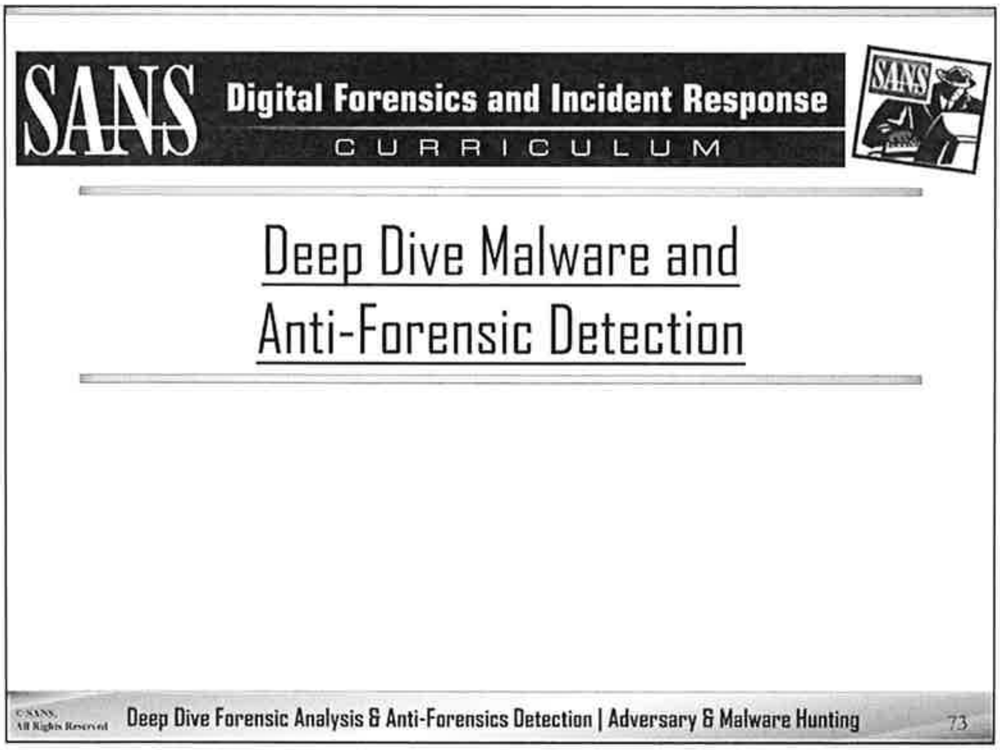
Deep Dive Forensic Analysis & Anti-Forensics Detection

Advanced Evidence of Execution
- `ShimCache`
- `RecentFileCache.bcf`
- `Amcache.hve`

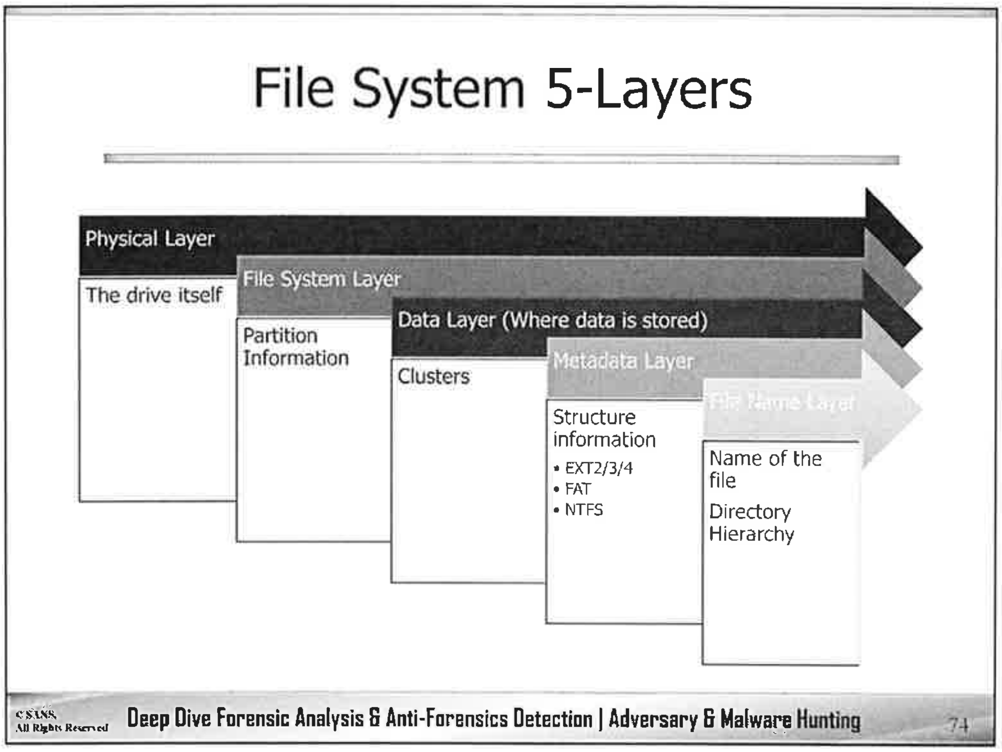
Historical Data Extraction
- Restore Point Analysis (WinXP)
- Volume Shadow Analysis (Win7/8 & Server 2008/2012)

Deep Dive Forensics & Anti-Forensics Detection
- Filesystem Based Analysis
- Sleuthkit Toolset
- Partition / Volume Analysis
- Data Layer Analysis
- Stream-Based Data Carving
- File-Based Data Carving
- NTFS Filesystem Analysis

This page intentionally left blank.



This page intentionally left blank.

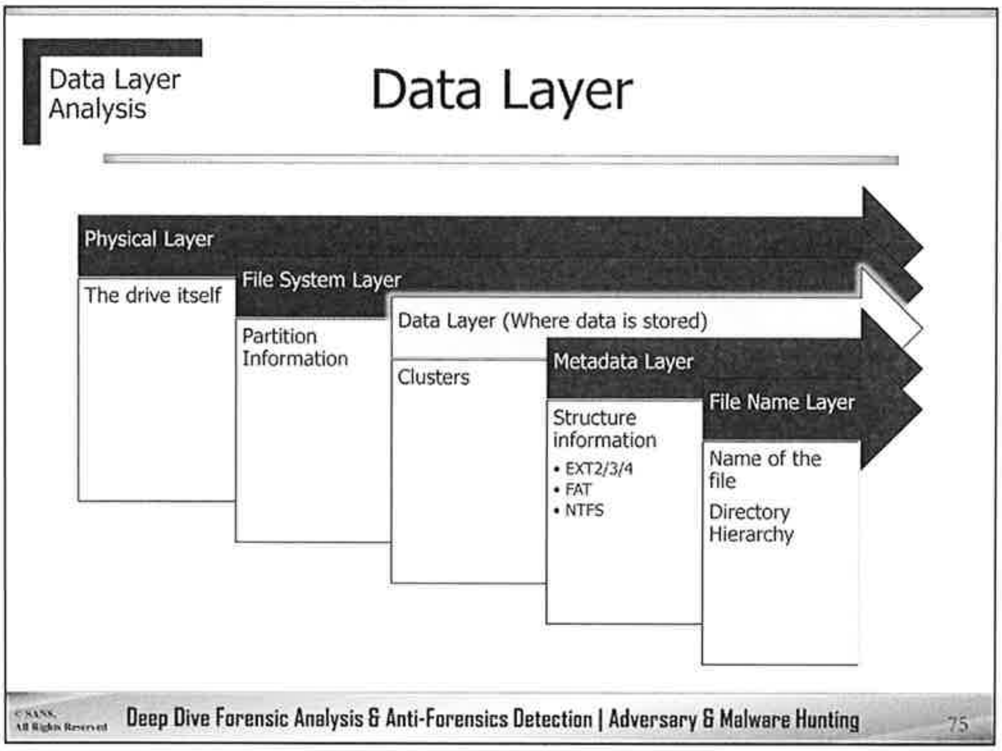


This is the overview of the file system 5 layers that we will be examining over the next few sections. We start with the physical and the partition details of a drive themselves which leads us to the actual file system. The file system will be made up of 3 layers (Data, Metadata, and File Name Layers). In order to understand how to best recover data during a case, we should understand how it is intentionally structured. Anti-Forensics techniques tend to overlook some of the more obvious structural foundations of a system because they are designed to fool a casual user, not someone familiar with the inner workings a file system forensics.

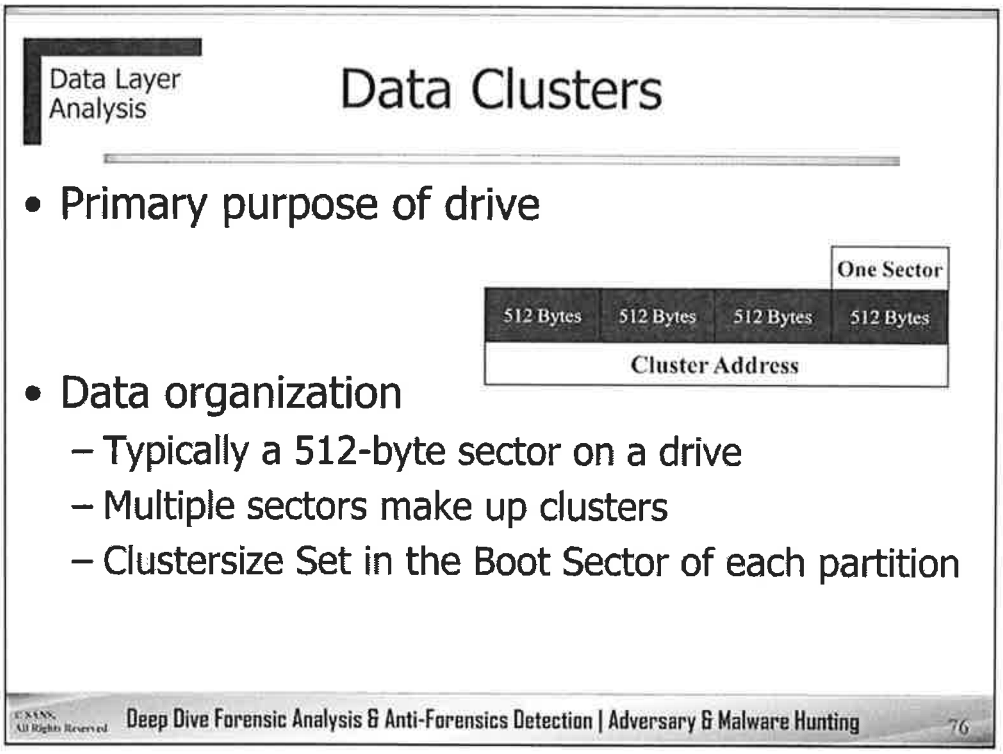
To extend that theory we will begin by breaking down the 5 layers initially.

From the 1st two layers we will examine the Physical layer; the File System layer will be used to discuss how a drive is initialized, partitioned, and then formatted. We will discuss the Master Boot Record, the partitions themselves, and how the Windows Operating system will identify a specific physical drive that is attached to it.

We then will discuss from a very high level how the 3 layers of the file system work together to actually store file data.



This page intentionally left blank.



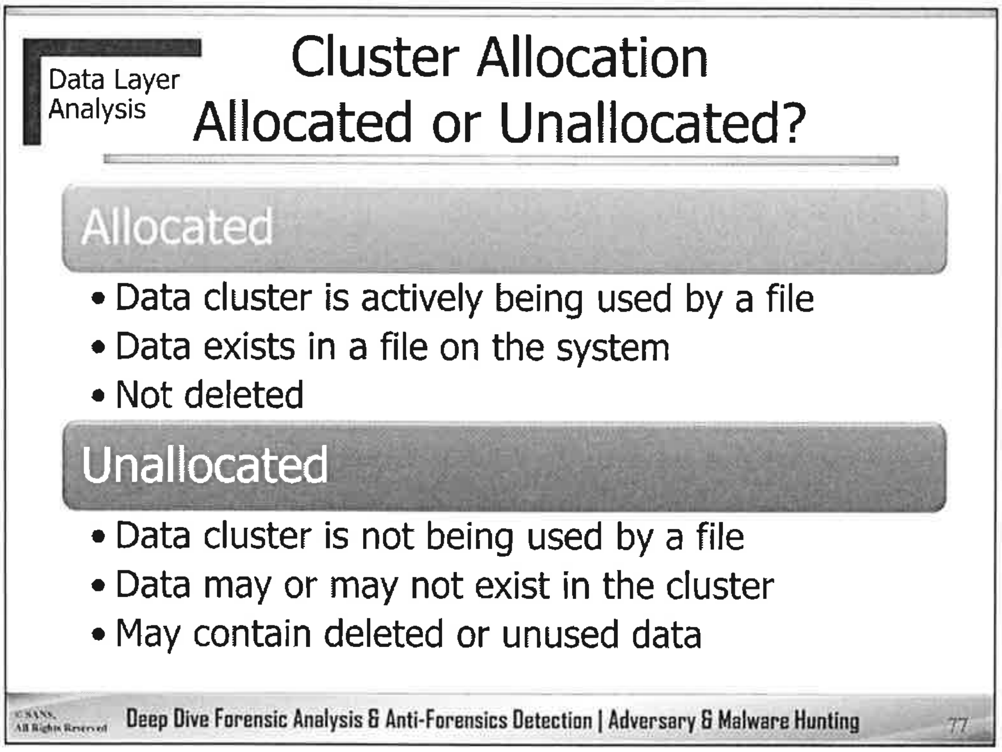
The Data Layer is the reason for having a file system. This is where file data is saved.

Most file systems use a unit called a sector that is 512 bytes in size. To make it efficient to allocate files in consecutive sectors, most file systems use larger data units that are consecutive sectors. The size of these data units ranges from 1 sector to 32 (or more) sectors.

These data units have different names in different file systems.

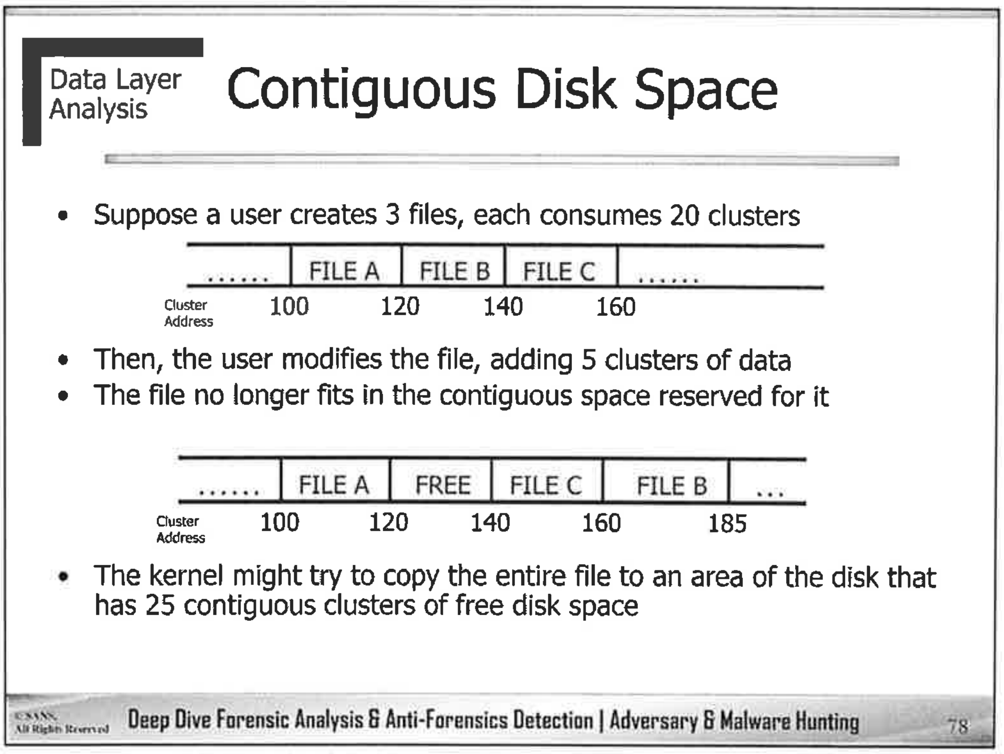
- FAT: Clusters
- NTFS: Clusters

Refer to pages 181-184 of your **Filesystem Forensic Analysis** book.



Data chunks will be in one of two states on the file system: used or not used. Each chunk of data (cluster) is either owned by an existing file or is waiting to be used. This is generally referred to as free space. Even though the space is free, it does not necessarily mean that it is free of data. Files that were deleted on the system could have written to these clusters at one point. This space is considered unallocated by the file system. Even though the space is unallocated, critical evidence can be recovered from these clusters despite not being recoverable by ordinary file recovery.

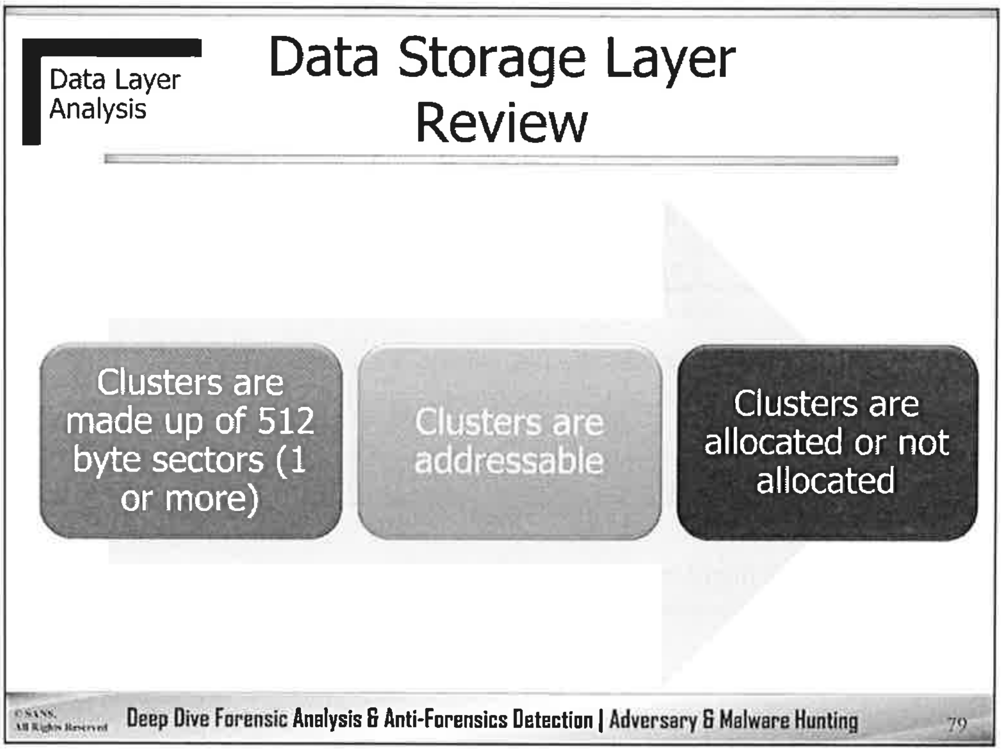
If a file is not fully recoverable, a piece of that file may still be recoverable. That piece of the file is called a file fragment. A fragment may be one or more clusters of data, but alone would not be the full file. For example, an e-mail found on a system may be recoverable, however, you may only obtain half of the e-mail. The other half was written over when the file system needed the data cluster that the e-mail portion resided in.



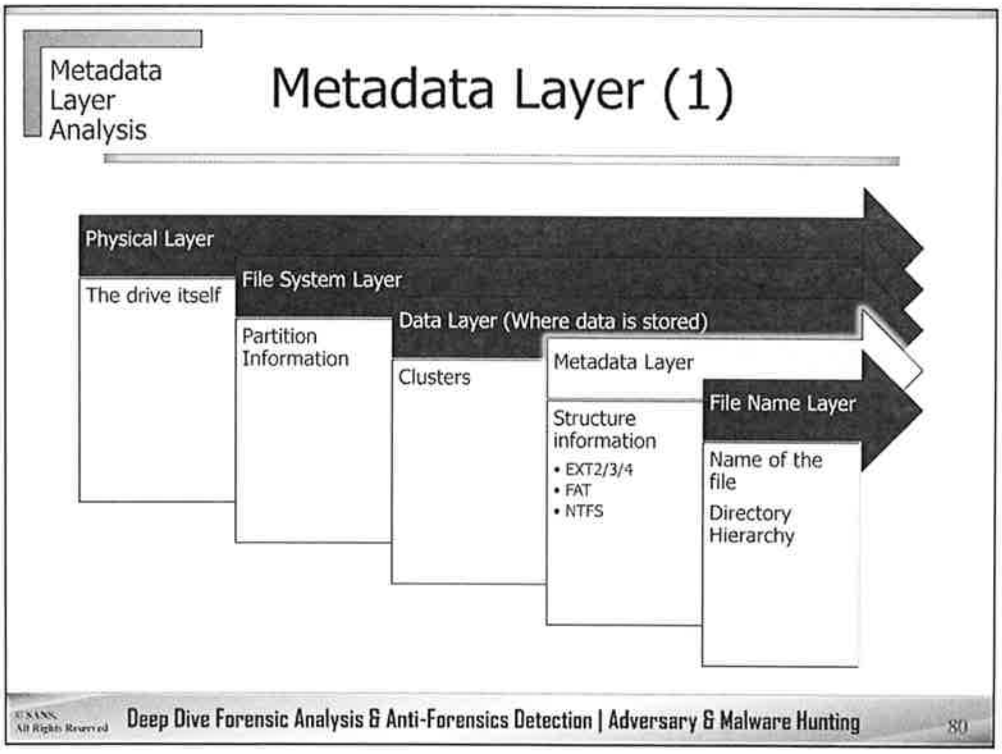
So then, given that a file is to be created, how is all of this laid out on disk? One might think that the file is written to disk in a contiguous manner. That is to say, if the file to be created is 5000 bytes, then the disk is searched for adjacent disk clusters that comprise 5000 bytes of storage. This is certainly the easiest way to do things, but in reality it doesn't work too well. Consider the example on the slide.

A user creates 3 files. Each file consumes 20 clusters of disk space. After some time, the user needs to modify file B by adding 5 clusters worth of data. This new file cannot be stored in the old location because there is not enough room. Rather than sliding the entire C file down 5 clusters, we might write file B after file C and leave the disk clusters previously occupied by file B – free (but the old data would still reside there).

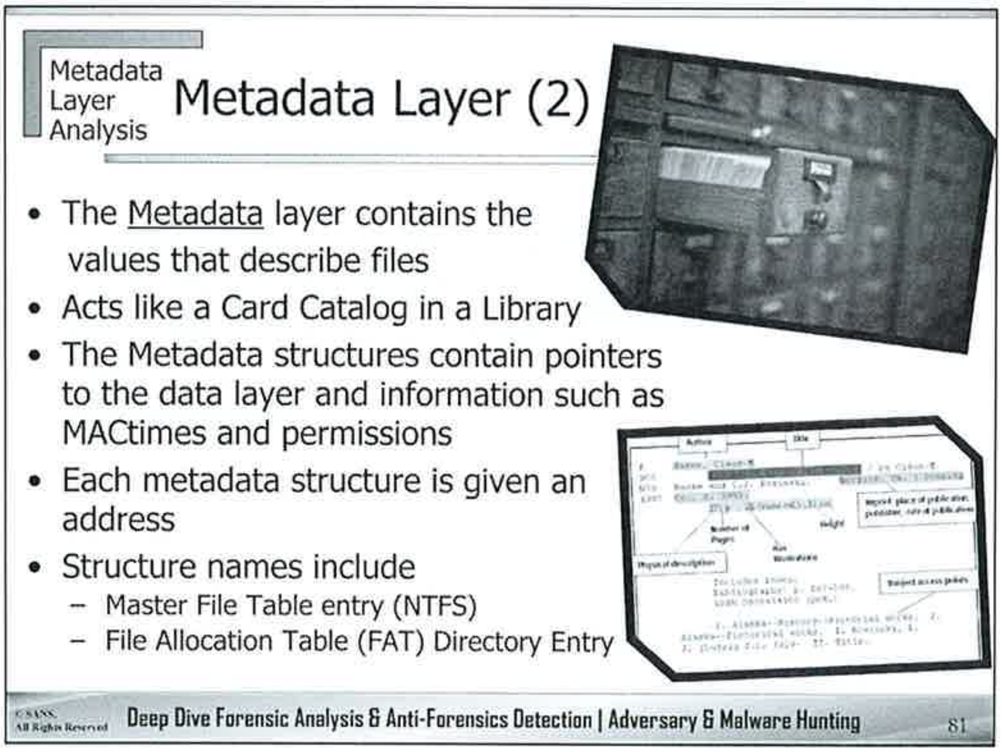
Most file systems will attempt to write data in contiguous clusters as the first option. Only when a file is too large to fit in contiguous clusters will the file system fragment a file. This is good for the forensic investigator as the majority of files that will be recovered will be contiguous rather than fragmented.



This is the essential information about the data layer that would be helpful to understand.



This page intentionally left blank.



Typical file systems store virtually all data in files. The most important of these are a set of special files, which are typically called *metadata structure* or *inodes*. The prefix "meta-" means self-referring. So "metadata structures" are structures that contain data *about* data. And that's exactly what these structures do. They contain internal information about the real data stored on the file system. For example, it could contain a listing of directory, timestamps, and file owners.

All file systems have some structure that is used to describe a file. The metadata layer contains those structures. These structures are called different things in different file systems:

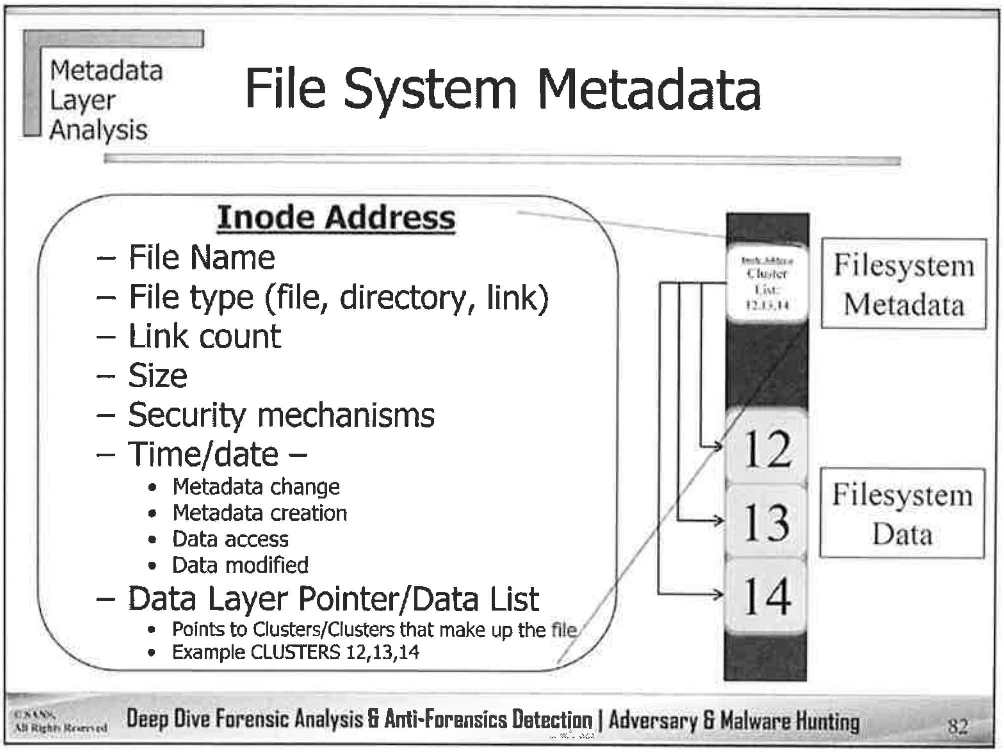
NTFS: Master File Table (MFT) entry

File Allocation Table (FAT): Directory Entry

These structures typically do not contain the actual name of the file. They contain descriptive information such as MACtimes, permissions, owner user id, and size. This layer also has some method of referring to the data units that have been allocated to the file. For Unix-based file systems, there are a series of pointers to the different fragments. For FAT, the File Allocation Table is used to find "Chains" of clusters.

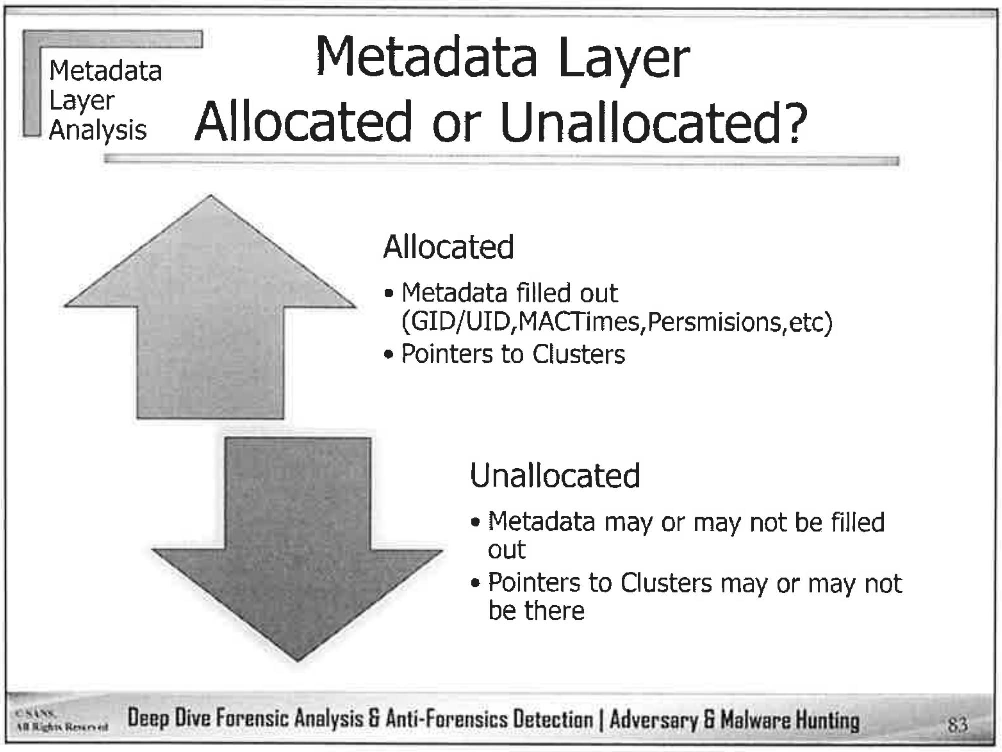
Each structure is given an address. We will use this address when referring to structures in this layer. This structure is typically hidden (especially for deleted files) from users, but there is a lot of useful information in them.

Refer to pages 186-190 of your **Filesystem Forensic Analysis** book.



The basics of any file system, Unix or Windows, are very similar, while the actual implementations are different. Reducing things to a common character and knowledge base will help an investigator know that similar techniques can be employed across a wide variety of file system types while only the specific tool that is used changes.

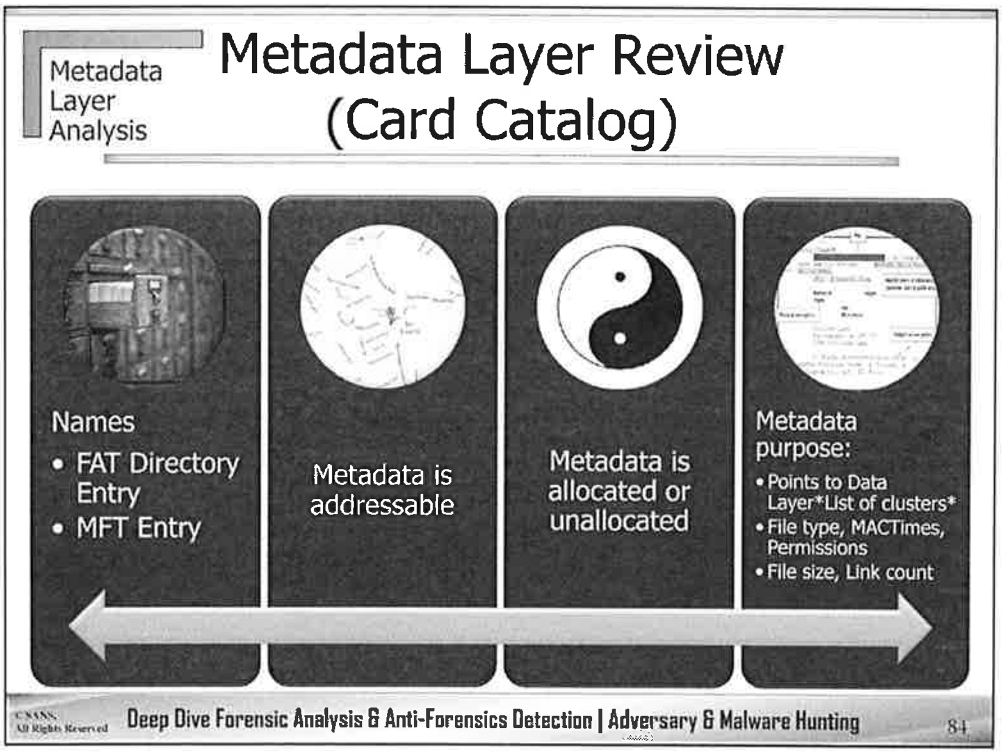
In most file systems, elements such as the name, the type, a pointer to the data, the data size, time data, and a security mechanism will exist. Some have more robust features (NTFS and ExFAT) than others, while some have less (FAT).



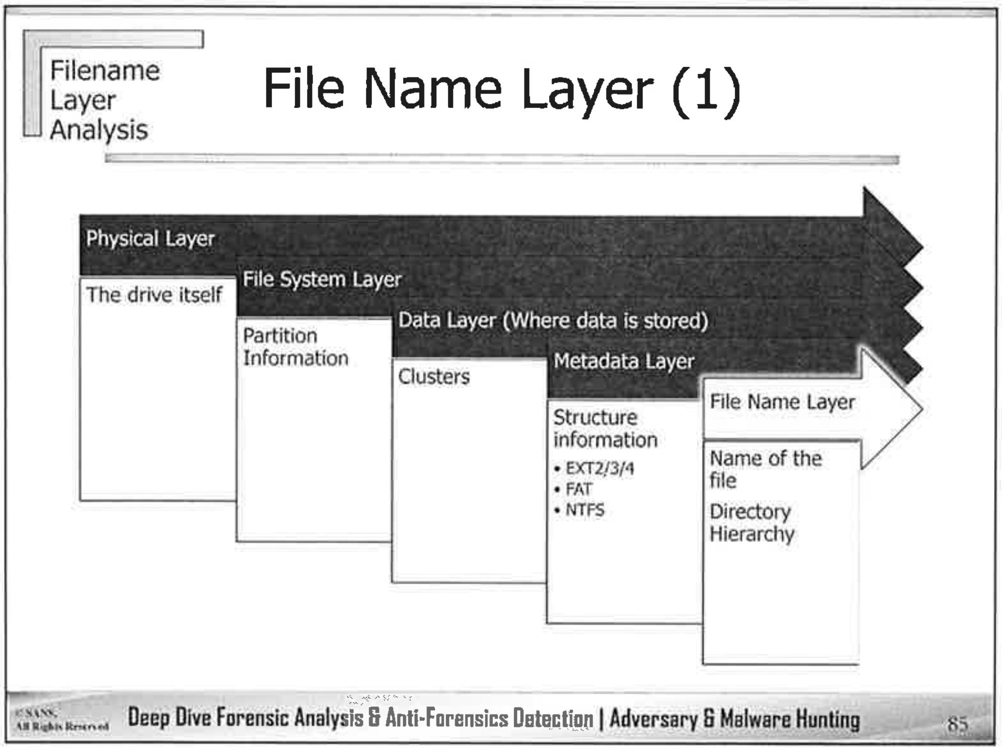
Like data blocks/clusters, inodes are also allocated or unallocated. An allocated inode is a file that is in use by the file system. A file with a name points to that inode structure in order to tell the operating system where the file data can be found. If an inode is unallocated, it was never written to, or it might contain the inode data of a file that was recently deleted.

If an inode was never in use by the file system, the data will never be filled out. When a file is deleted the inode data is rarely wiped or overwritten.

Refer to pages 195 of your **Filesystem Forensic Analysis** book.



This is the essential information about the metadata layer that would be helpful to understand.



This page intentionally left blank.

Filename Layer Analysis

File Name Layer (2)

File names potentially stored in two places:

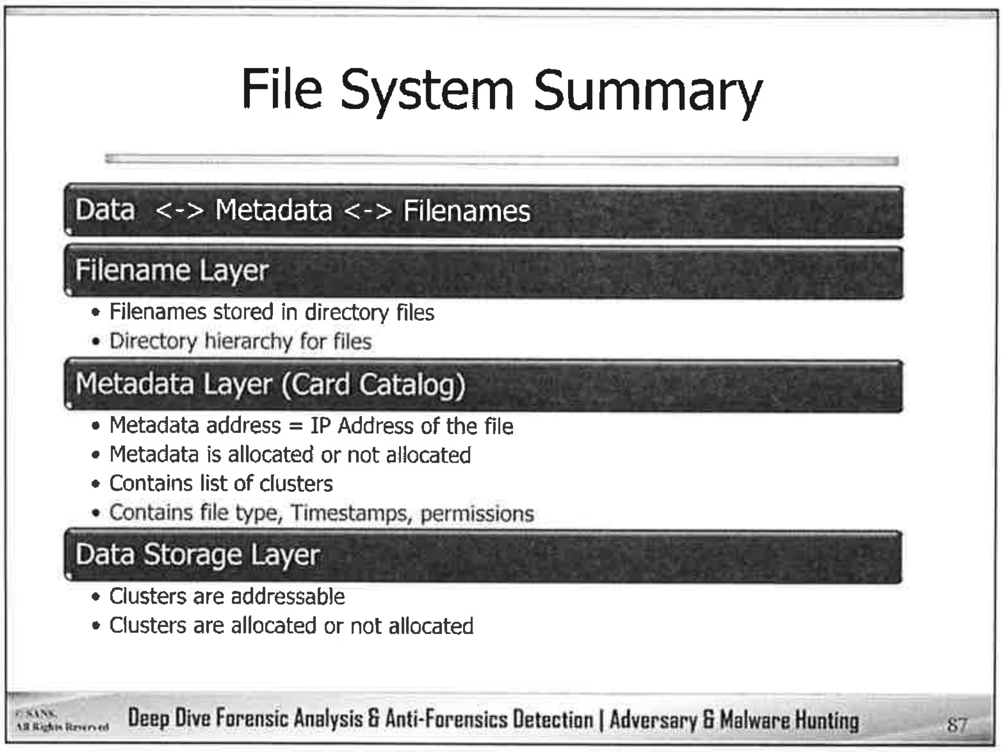
- File Metadata
  - MFT Entry, FAT Directory Entry *Windows*
  - Unix does not store filename in metadata
- Directory File
  - Contains list of files/directories in that directory

Filenames point to the Metadata Address

- Similar to Domain Names that point to specific IP Addresses

The File Name layer is typically a separate structure that gives names to files. The metadata layer can describe everything about a file, but it is often inconvenient to have to remember that /etc/password is inode 312. The file name structures are typically stored in the data units allocated to the parent directory. They contain the name of the file and the address of the metadata structure (except in FAT because everything is stored in the parent directory).

When files are deleted, the file system will hide the file name from the user, but much data can be recovered using forensic tools.



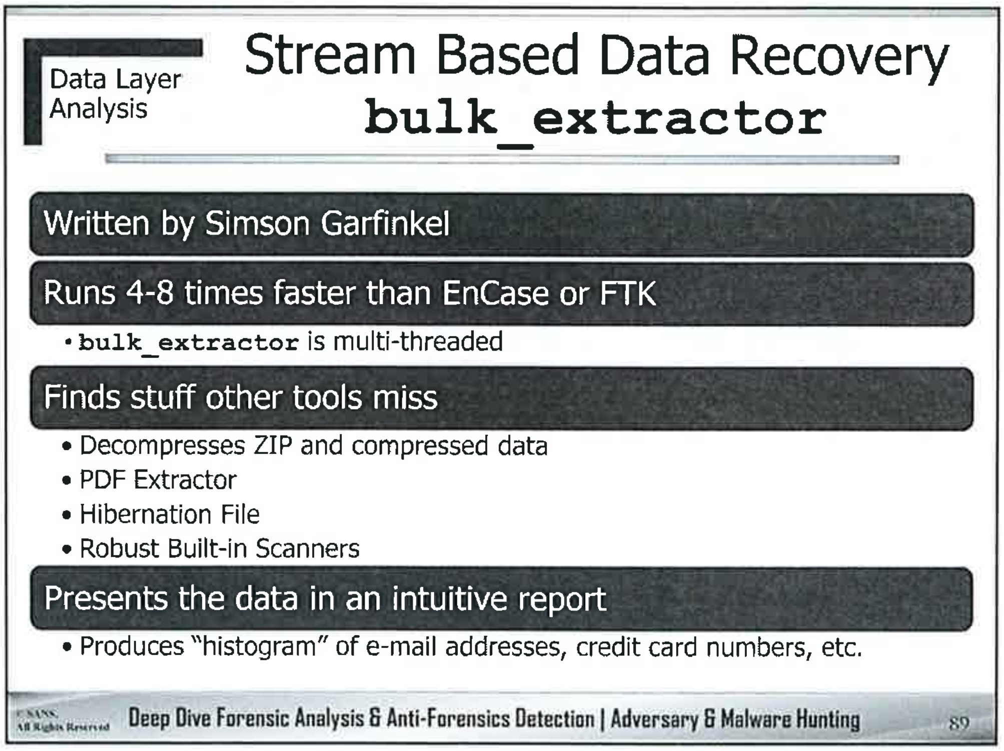
This page intentionally left blank.

SANS Digital Forensics and Incident Response
CURRICULUM

Deep Dive Forensics:
String and Stream-based Data Carving

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 88

First we will focus on the file system layer tools.



Data carving and extraction is sometimes a very difficult task and there are several approaches that can be utilized to extract relevant data from an image. The **first** is the filesystem method – using the metadata and the data structures to piece together files in their original format. The requirement of filesystem based extraction is that your tools must be configured to parse the internal data of the filesystem itself. The **second** method is stream based data extraction. In this method, data carving is performed against data chunks without any recognition of the data format or the filesystem. The benefit to the second method is that you will be able to examine and extract all data (deleted or existing) from an image without missing anything. In addition, if you are carving out data based on a stream examination method then you can point your examination tools against raw data sources such as memory images, hibernation files, and pagefiles.

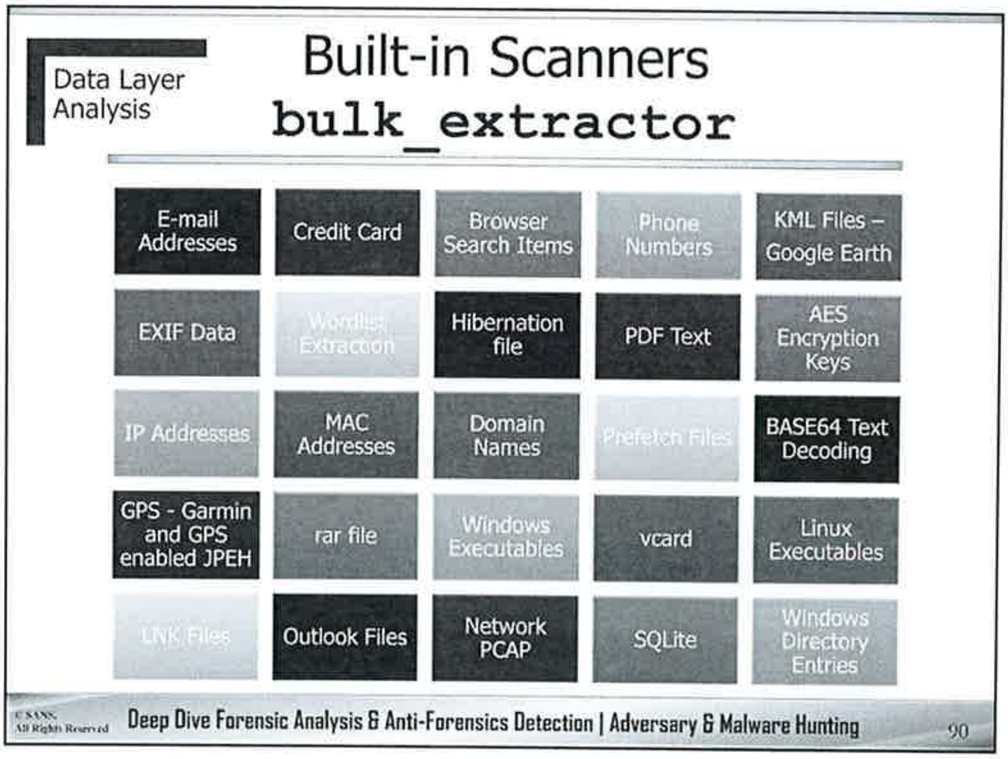
We will discuss both in this class. We will first start with stream based data extraction and finish with filesystem based data extraction. However, even though both are different, stream based extraction can also be used to aid in filesystem based data recovery. In order to begin, we will utilize a tool specifically optimized for stream based data recovery, bulk_extractor.

Nothing quite describes bulk_extractor like its author can, below is a snippet from the bulk_extractor paper released with the tool.

"**bulk_extractor** is distinguished from other forensic tools by its speed and thoroughness. Because it ignores file system structure, **bulk_extractor** can process different parts of the disk in parallel. In practice, the program splits the disk up into 16MiByte pages and processes one page on each available core. This means that 24-core machines process a disk roughly 24 times faster than a 1-core machine. **bulk_extractor** is also thorough. That's because **bulk_extractor** automatically detects, decompresses, and recursively re-processes compressed data that is compressed with a variety of algorithms. Our testing has shown that there is a significant amount of compressed data in the unallocated regions of file systems that is missed by most forensic tools that are commonly in use today."[1]

Citations:

[1] Bulk Extractor Home Page: http://afflib.org/software/bulk_extractor



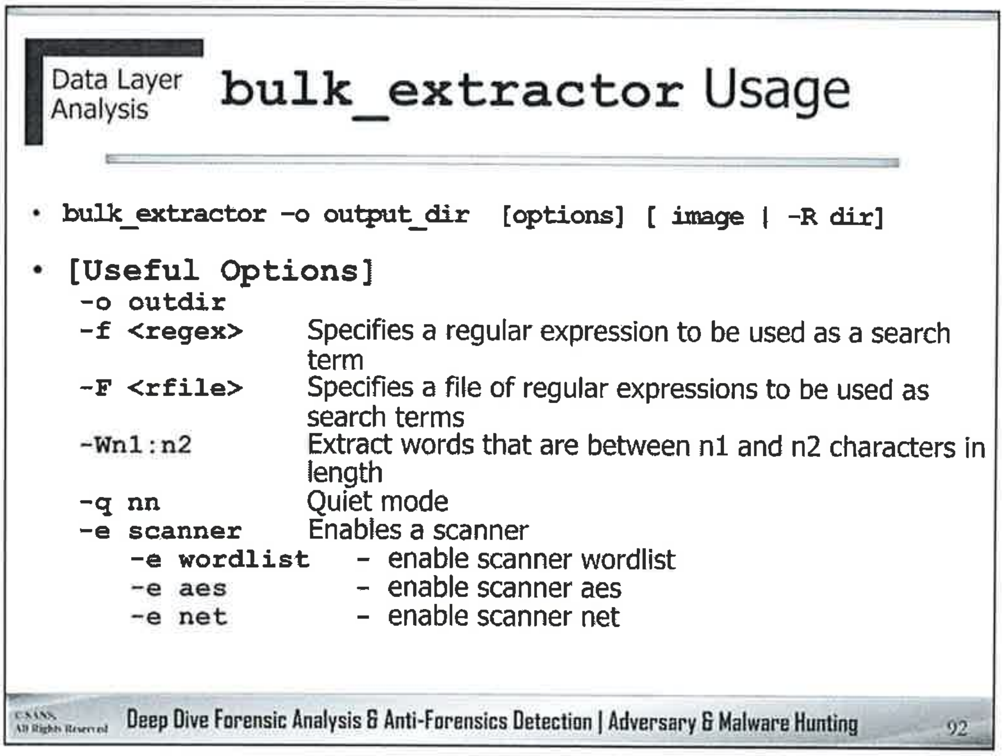
One of the greatest benefits of bulk_extractor are the built in data carvers looking for specific types of data out of a stream. The power of bulk_extractor extends beyond just a simple keyword search as data could exist in many forms inside the data of an image. For example, a zip file or webmail fragment could exist in the stream compressed. Bulk_extractor will read the compressed data and extract the readable text from it as well.

If you are using a keyword list, you can easily pull out regular expressions from an entire image just like using the grep and srch_strings command combination. However, there are many internal preprocessors that will also extract specific data that could be of interest to the examiner. The data found in TCP connections, domain names, GPS coordinates, and more are all automatically collected and stored for later parsing once bulk_extractor is complete.

Current list of bulk extractor data stream scanners:

- scan_accts - Looks for phone numbers, credit card numbers, and other numeric info.
- scan_aes - Detects in-memory AES keys from their key schedules
- scan_base16 - decodes hexadecimal test
- scan_base64 - decodes BASE64 text
- scan_elf - Detects and decodes ELF headers
- scan_exif - Decodes EXIF headers in JPEGs using built-in EXIF parser.
- scan_exiv2 - Decodes EXIF headers in JPEGs using libexiv2 (for regression testing)
- scan_email - Scans for e-mail addresses, URLs, and other text-based information.
- scan_exif - Decodes EXIF headers in JPEGs using built-in decoder.
- scan_find - keyword searching scan_facebook- Facebook HTML

- scan_gps - Detects XML from Garmin GPS devices
- scan_gzip - Detects and decompresses GZIP files and gzip stream
- scan_hashdb - Search for sector hashes/ make a sector hash database
- scan_hiber - Detects and decompresses Windows hibernation fragments
- scan_httplog - search for web server logs
- scan_outlook - Decrypts Outlook Compressible Encryption
- scan_json - Detects JavaScript Object Notation files
- scan_kml - Detects KML files
- scan_lightgrep - performs searches with LightBox Technology's LightGrep.
- scan_net - IP packet scanning and carving
- scan_pdf - Extracts text from some kinds of PDF files
- scan_sqlite - SQLite3 databases (only if they are contigious)
- scan_rar - RAR files
- scan_vcard - Carves VCARD files
- scan_windirs - Windows directory entries
- scan_winlkn - Windows LNK files
- scan_winpe - Windows executable headers.
- scan_winprefetch - Extracts fields from Windows prefetch files and file fragments.
- scan_wordlist - Builds word list for password cracking
- scan_xor - XOR obfuscation
- scan_zip - Detects and decompresses ZIP files and zlib streams



From the bulk_extractor help and manual pages:

bulk_extractor -o output_dir [options] [image | -R dir]

DESCRIPTION

bulk_extractor scans a disk image (or any other file) for a large number of pre-defined regular expressions and other kinds of content. These items are called features. When it finds a feature, bulk_extractor writes the out-put to an output file. Each line of the output file contains a byte offset at which the feature was found, a tab, and the actual feature. Features therefore cannot contain the end-of-line character. bulk_extractor includes native support for EnCase (.E01) and AFFLIB (.aff) files, if it is compiled and linked on a system containing those libraries. Alternatively, the -R option can be used to recursively scan and process a directory of individual files (disk images in such a directory will be treated as files, not as disk images).

bulk_extractor is multi-threaded. By specifying the -j option, multiple copies of the program can be run. Each thread writes its results into its own feature file. The files are then combined by the primary thread when all of the secondary threads complete.

bulk_extractor is a two-phase program. In phase 1 the features are extracted. In phase 2 a histogram is created of relevant features. bulk_extractor will also create a wordlist of all the words that are found in the disk image. This can be used as a dictionary for cracking encryption.

The options are as follows:

-o outdir

Specifies the output directory, which will be created by bulk_extractor.

-b bannerfile.txt

Read the contents of bannerfile.txt and stamp it at the beginning of each output file. This might be useful if you have some kind of privacy banner that needs to be stamped at the top of all of your files.

-r alert_list.txt

Specifies an alert list, (or red list), which is a list of terms that, if found, will be specifically flagged in a special alert file that begins with the letters ALERT. The alert list may contain individual terms, which must be found in their entirety and are case-sensitive, or wildcards with standard Unix glob-bing (e.g. *@company.com). Globbed terms are case-insensitive.

-w stop_list.txt

Specifies a stop list, (or white list), which is a list of terms that, if found, will be placed in a special stopped file (rather than in the main file). The whitelist may also contain globbed terms.

-s context_stoplist.txt

Specifies a context-dependent stoplist, which is a list of tokens to be stopped (but only when they have a particular context).

-p path/format

Open a disk image and print the information found at path. The format specification may be r for raw out-put and h for hex output.

-F <rfile>

Specifies a file of regular expressions to be used as search terms.

-f <regex>

Specifies a regular expression to be used as a search term.

-q nn

Quiet mode. Only prints every nn status reports.

These commands are useful for tuning operation:

-C n

Specifies the size of the context window.

-g n

Specifies the size of the margin in bytes.

-Wn1:n2

The scan_wordlist scanner should only extract words that are between n1 and n2 characters in length.

-j n   Use n threads for analysis. Normally you do not need specify this, as the default is the number of processors on the current computer.

The following commands are useful for debugging:

-V   Print the version number

-R outdir

Restarts the program from where it left off for a particular directory.

-B nn

Set the dedup Bloom filter to nn bits. This is used by the scan_wordlist scanner.

-M nn

Specifies a maximum recursion depth of nn.

-z pagenum

Start on page number pagenum.

-Y offset

Start at offset

-dN

Enable debugging level N.

Finally, you can control scanners with these options:

-P <dir>

Specifies a directory in which to find plugins.

-E scanner

Turns off all scanners, then enabled scanner .

-e scanner

Enables a scanner.

-e wordlist - enable scanner wordlist

-e aes - enable scanner aes

-x scanner

Disables a scanner.

-x accts - disable scanner accts

-x base64 - disable scanner base64

-x kml - disable scanner kml

-x email - disable scanner e-mail

-x gps - disable scanner gps

-x net - disable scanner net

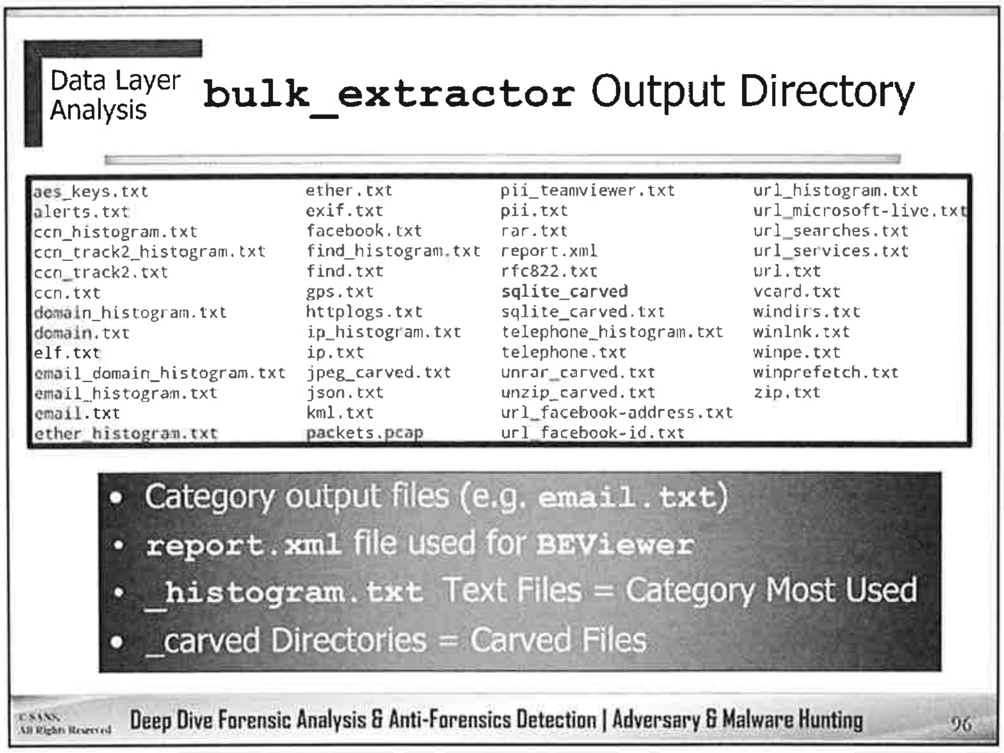
-x exif - disable scanner exif

-x zip - disable scanner zip

-x gzip - disable scanner gzip

-x pdf - disable scanner pdf

-x hiber - disable scanner hiber



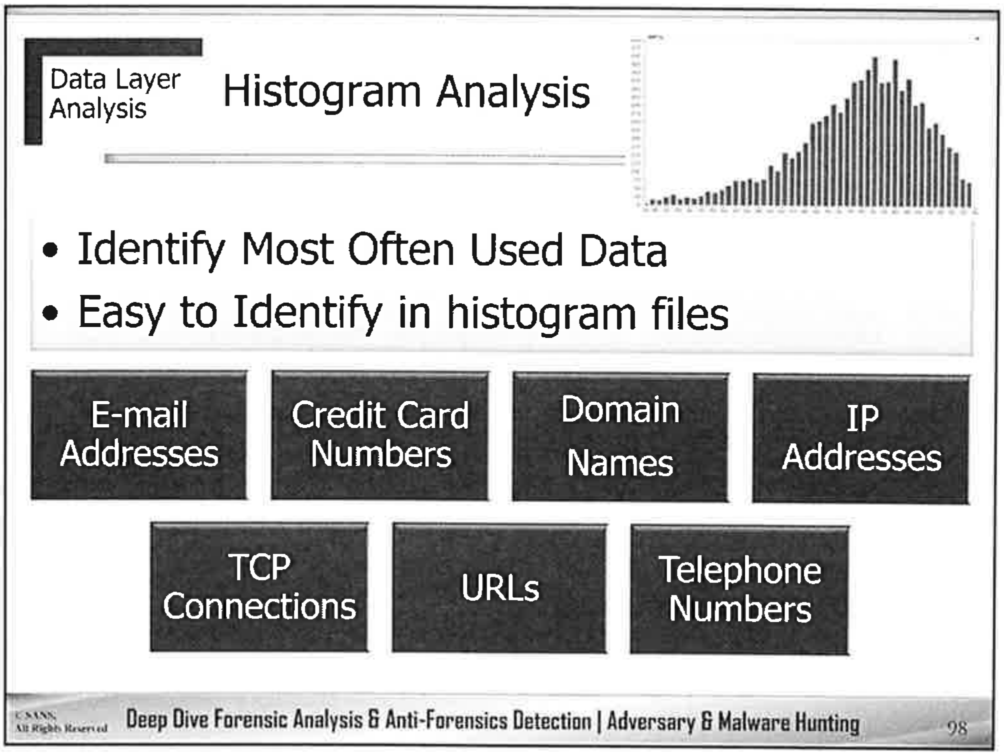
The following are the descriptions of the fields found in the bulk_extractor output directory. Each reports file will not only contain the data described, but the byte offset in the image where the data was found.

- aes_keys.txt - AES encryption keys alerts.txt - Features found on alert list (redlist)
- ccn.txt - credit card numbers
- ccn_track2.txt - Track 2 information
- domain.txt - All extracted domain names and IP addresses
- email.txt - extracted e-mail addresses
- ether.txt - extracted Ethernet addresses. Currently overcollecting due to a failure to consider local context.
- exif.txt - All exif fields from JPEGs; extracted as XML.
- find.txt - Hits on find command. hex.txt - Notable hexadecimal constants
- gps.txt - Extracted GPS coordinates from Garmin XML and GPS-enabled JPEG files
- ip.txt - Extracted IP addresses from scan_net cksum-bad indicates checksum test failed; those are less likely to actually be IP addresses.
- json.txt - Extracted and validated JavaScript Object Notation fragments.
- kml.txt - Extracted KML files
- rar.txt – Extracted .rar archive files
- report.xml – The DFXML file that explains what happened.

- rfc822.txt - All extracted RFC822 headers
- tcp.txt - Summaries of all extracted UDP and TCP packets.
- telephone.txt- Extracted phone numbers
- url.txt - Extracted URLs
- url_facebook-id - extracted Facebook IDs
- url_microsoft-live - extracted Microsoft Live IDs
- url_searches - extracted search terms
- url_services - extracted services from URLs
- winprefetch.txt - Windows prefetch files and fragments, recoded as XML for easy processing.
- wordlist.txt - All the words
- zip.txt - Information about all ZIP files and zip components.

Current list of carving directories:

- unrar/
- jpeg/
- unzip/
- sqlite/



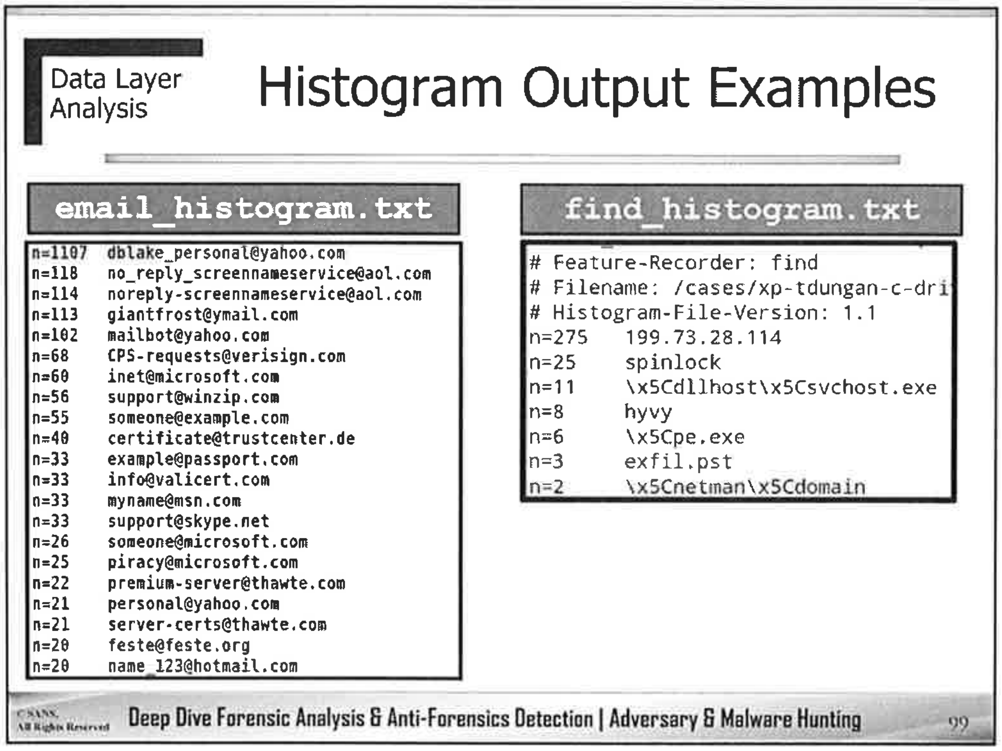
One of the most useful bulk_extractor output files are the histogram files. Histograms are used to plot density of data. In this case, it will show the items that appear most often in the extracted output files. The histogram files will perform a unique and a total of all the fields in a particular category. Using this information you can tell a great deal of information about the computer system and even the habits and preferences of the main user of the system.

The following fields will commonly contain histogram data created through the use of bulk_extractor (e-mail addresses, credit card numbers, domains, IP addresses, TCP connections, URLs, and telephone numbers.

For example, the primary user will probably have their e-mail address show up more times than any other. The people they receive and send e-mails to would show up in the output next. Using this information you can start to perform link analysis based on the usage and existence of certain data types found in the system over other ones.

What if this system was used to connect repeatedly to another system in your infrastructure due to an APT infestation? Histogram analysis might show latent packets still showing up in the hibernation file or memory image.

Histogram analysis could be very telling in a situation where you might not know where to look. What if a user of the machine faked an identify of another user online. The 2nd user id would show up repeatedly in the histogram analysis.



This is an example output from a bulk_extractor run from the image we used in FOR408: Windows Forensics In-Depth. In the first example, we can see the primary user of the machine was Donald Blake and the e-mail address he used the most dblake_personal@yahoo.com was found 1,107 times in the image.

In the find_histogram.txt, we find many of the hits for the specific keywords that we found in our FOR508 XP-tdungan-c-drive image. This shows the number of references to files in the system. The character \x5c is a "\" backslash.

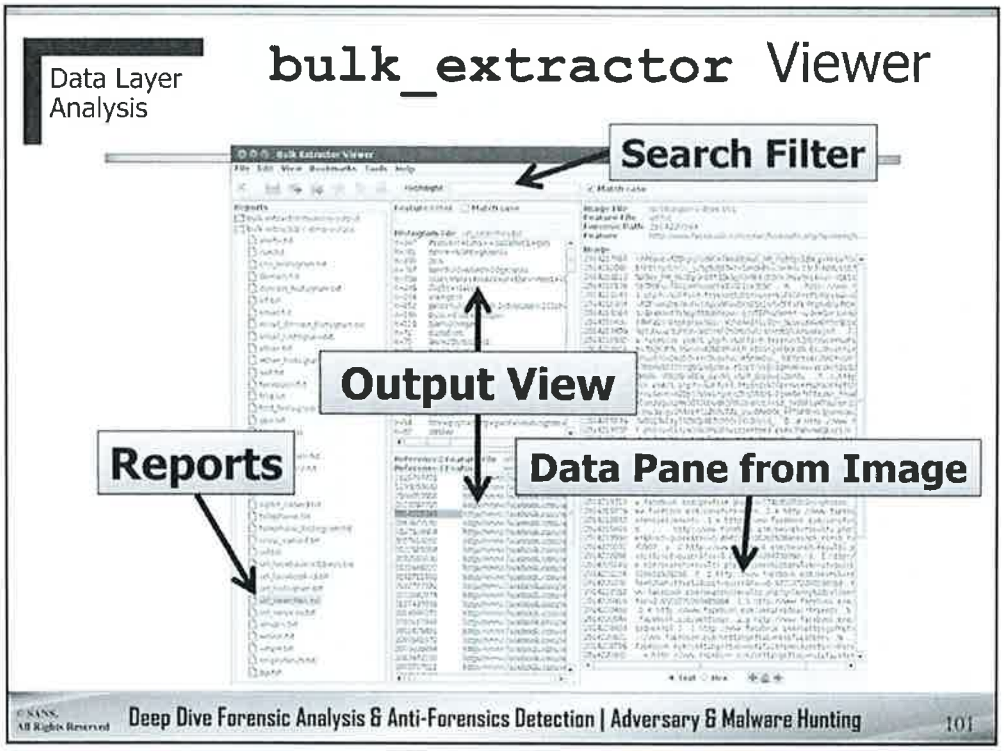
In the end, histogram analysis tends to be very useful in looking for trends in data on a single system. To that end, bulk_extractor's capability of automatically creating this data for you, makes the tool twice as valuable in the end.

tcp_historgram.txt

```
n=31    20.255.178.252:847 -> 1.0.198.192:46846 (TCP)
n=25    0.106.37.95:23179 -> 102.59.199.117:52968 (TCP)
n=9     7.90.102.193:13311 -> 108.5.218.9:18387 (TCP)
n=8     0.144.15.18:17611 -> 4.203.15.22:2063 (UDP)
n=7     0.80.255.181:36237 -> 232.253.255.255:61693 (TCP)
n=7     137.69.212.139:1016 -> 69.224.139.56:35651 (TCP)
n=7     88.16.0.0:49632 -> 209.227.139.195:1104 (TCP)
n=7     88.16.0.0:51137 -> 193.231.2.139:57348 (TCP)
n=5     70.2.2.198:17933 -> 70.12.4.198:1160 (TCP)
n=4     117.236.139.206:15299 -> 87.255.80.64:35141 (TCP)
n=4     196.139.69.196:34692 -> 59.69.36.15:0 (UDP)
n=4     221.179.70.249:63172 -> 248.200.76.255:17407 (TCP)
n=4     255.144.12.1:3972 -> 0.0.133.192:52224 (TCP)
n=4     255.80.96.132:30188 -> 192.116.30.255:35790 (TCP)
n=4     8.0.0.141:22358 -> 77.224.81.87:65360 (TCP)
n=3     0.0.235.218:29709 -> 131.125.36.0:50950 (TCP)
n=3     0.106.37.95:23691 -> 102.59.199.117:52968 (TCP)
n=3     117.228.255.117:4155 -> 248.86.255.80:50057 (TCP)
```

email_historgram.txt

```
n=1107  dblake_personal@yahoo.com
n=118   no_reply_screennameservice@aol.com
n=114   noreply-screennameservice@aol.com
n=113   giantfrost@ymail.com
n=102   mailbot@yahoo.com
n=68    CPS-requests@verisign.com
n=60    inet@microsoft.com
n=56    support@winzip.com
n=55    someone@example.com
n=40    certificate@trustcenter.de
n=33    example@passport.com
n=33    info@valicert.com
n=33    myname@msn.com
n=33    support@skype.net
n=26    someone@microsoft.com
n=25    piracy@microsoft.com
n=22    premium-server@thawte.com
n=21    personal@yahoo.com
n=21    server-certs@thawte.com
n=20    feste@feste.org
n=20    name_123@hotmail.com
```



bulk_extractor has a front-end GUI called BE Viewer that can be executed to not only view the data from bulk_extractor's report.xml file but can be used to actually run the tool itself.

In the above window, you can see that when you run BEViewer and select TOOLS-> Run bulk_extractor it will open a 2nd window in which you can select your image file, output directory, and the various options that can be used to direct the tool to your specific wishes without making command line mistakes.

It is also very useful as we can easily identify and see the various scanners built into the bulk_extractor interface. Personally, I prefer to build the execution of the tool via the GUI to avoid mistakes of having to re-run the tool later on if I forgot to add the correct option.

The BEViewer tool should open the reports.xml file. You would also use the image option to point it at the data image that bulk_extractor was originally run against to provide contextual relationships from the original image.

Reports Pane:

The various output reports from bulk_extractor are found here. Simply click on one of the reports (sometimes called features) to view the contents of that report in the output view.

Output View:

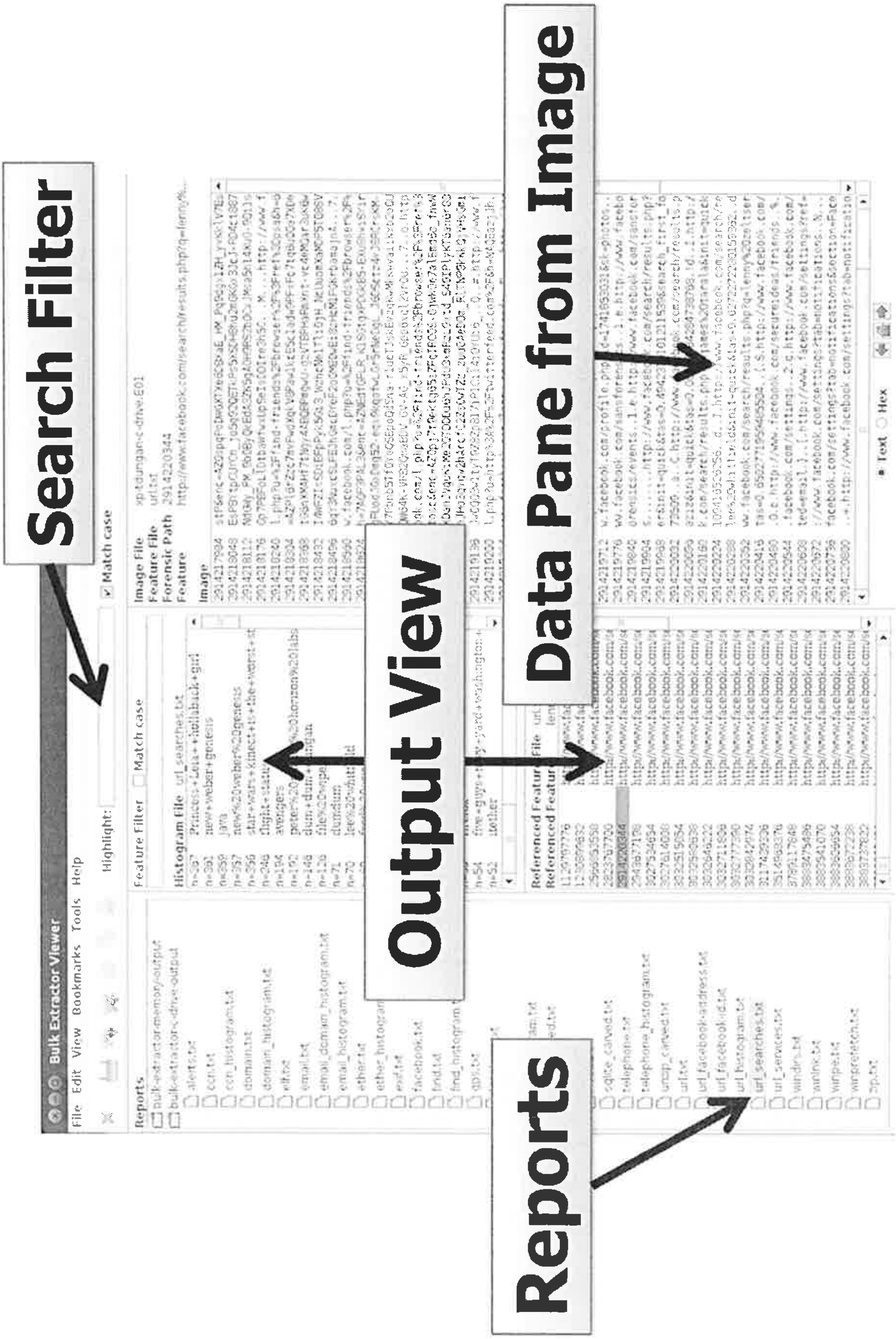
This is where you will see the data from the reports files and the respect byte offset for the data that they contain. If you select a term, it will highlight it yellow and identical terms will also appear in yellow.

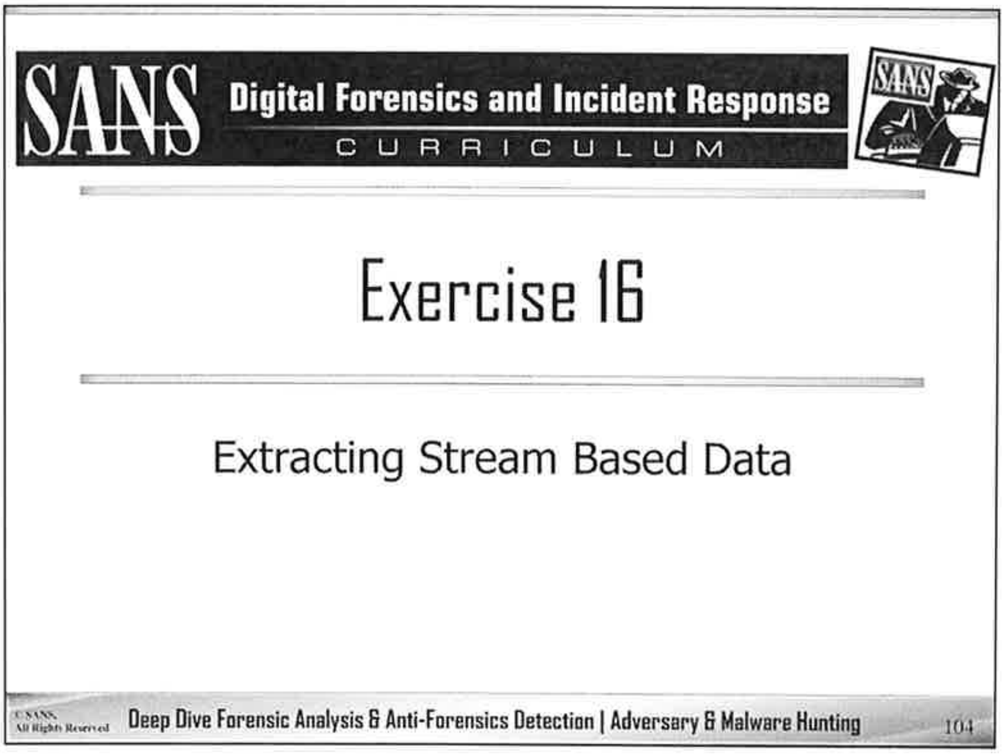
Search Filter:

If you are looking for a specific term in a report file, you can use the search filter to narrow your results. There is a case sensitive toggle button also found on this feature of the BEViewer.

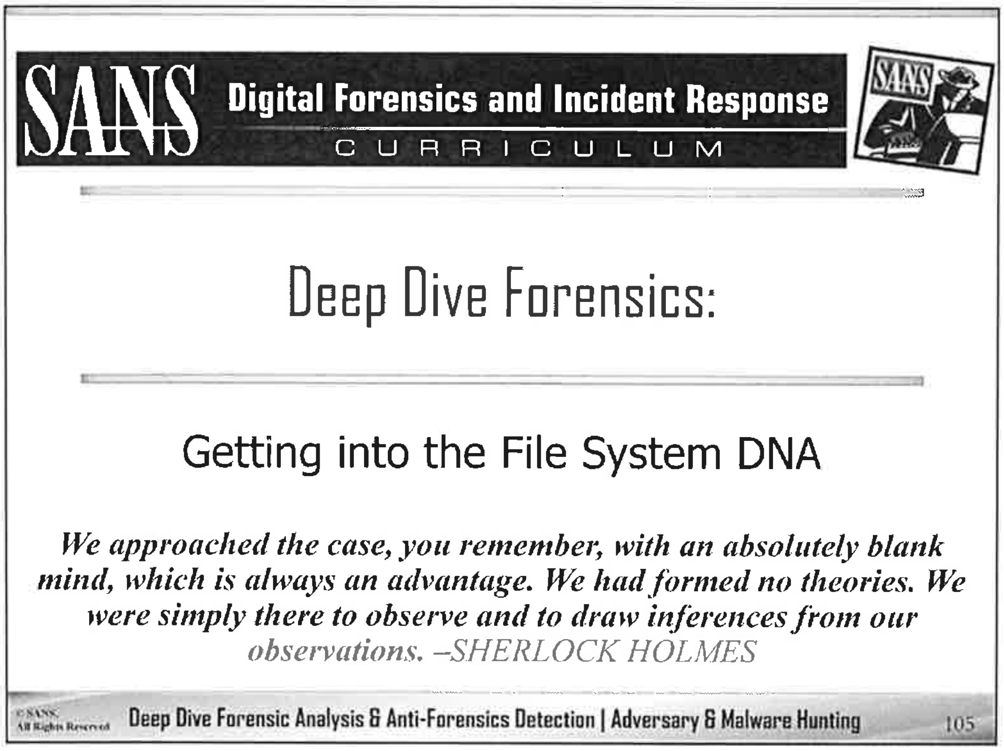
Data Pane From Image:

The data pane from the image will show the original location of the data in the image file that was parsed using bulk_extractor. This is very useful to provide some sort of context surrounding the data that you are viewing in the output view.

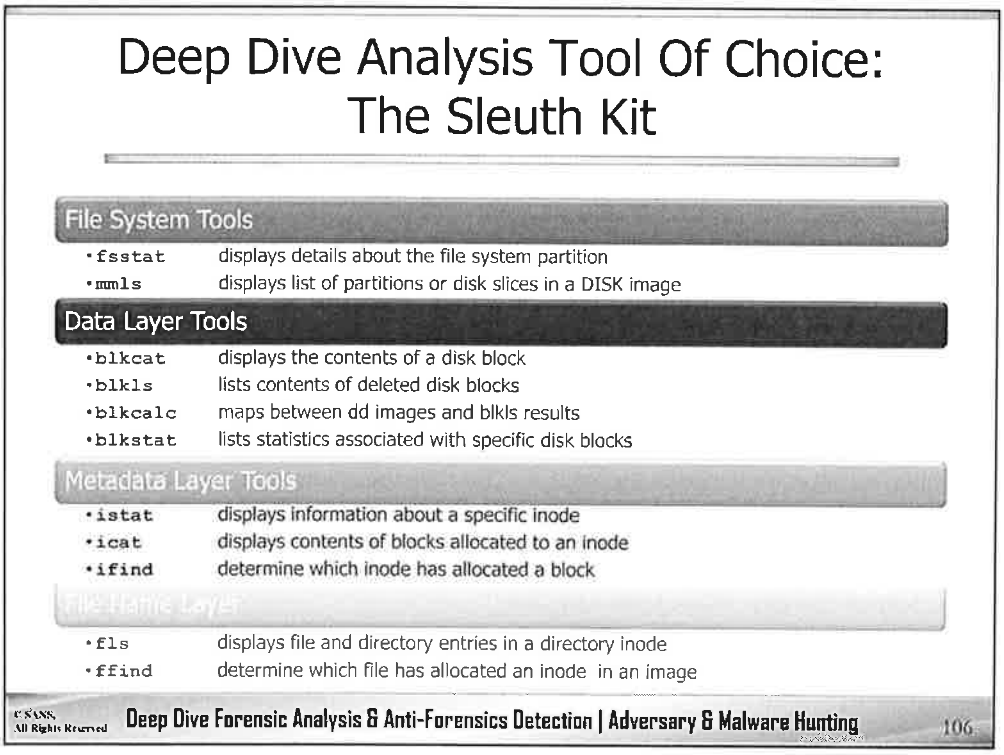

Search Filter
Output View
Data Pane from Image
Reports



This page intentionally left blank.



Now that we've laid all of the evidence out on the table, we can begin to establish a timeline of the hacker's activities on the system. All in all, there was very little time spent on the system.



Overview:

- Written by Brian Carrier
- We will now show the tools in The Sleuth Kit
- Works on both UNIX and WINDOWS platforms
- Some of these tools are also in TCT, but the Sleuth Kit version has more options
- There are a total of 27 tools in The Sleuth Kit
- They are based on the Unix model of small specialized tools
- They are organized by file system layers and the first letter of each tool name corresponds to the layer.
- The remainder of the name are standard across layers: stat, ls, find
- All commands take at least the image name and typically an address as well
- Can use disk or partition image files
- Can have single or split files
- The offset option is used to mark the beginning of the partition. Pulled from `mmls` output.
- Can use raw, EnCase, or AFF disk images

For the file system analysis section, we will use The Sleuth Kit. As The Sleuth Kit is based on the code from TCT, some of the tools in The Sleuth Kit can also be found in TCT. The Sleuth Kit tools are more flexible and provide more features, including support for FAT and NTFS and other features that will be shown.

There are 27 total tools in The Sleuth Kit. Some take a file system as input and provide output data and others run on the output data to present the data in a different format. For example, the mactime tool takes data from other tools to generate a timeline of activity. Similarly, the 'sorter' tool calls other tools to sort data by type.

Having to learn over 20 different tools sounds daunting, but the file system tools are organized by file system and data layers. There are a total of 5 layers in a basic model (shown next) and the first letter of each tool is based on the layer. The remainder of the name uses standard names, such as find, stat, and ls.

Let's go into the file system layers in detail.

There are eleven programs in the Sleuth Kit package that do various things according to the conceptual model discussed earlier.

The 'cat' tools display data (like the 'cat' tool in Unix). The 'stat' tools display statistics about an address, the 'ls' tools list details about many addresses, and the 'find' tools map between different layers.

The tools for the File System Layer start with 'fs'.

The tools for the data layer start with 'blk'.

The tools for the metadata layer start with 'i'. This is because the original TCT tools were designed only for Unix inodes.

The Human Interface layer all start with the letter 'f'. The 'fls' tool allows us to list files like the 'ls' command does in Unix, except it will also show us deleted files.

Some file systems have journals that record what changes are made to them and the journal layer tools will help show the journal contents. These tools start with 'j'.

The media management layer tools show the media management configuration and we already saw 'mmls' being used to list the partition table.

The disk layer shows the configuration of an actual hard disk. These tools are used to help determine if there is hidden data on a disk and remove it.

The remaining tools do not directly process a file system or disk image:

- Hash database tools
- The 'file' tool and 'sorter'
- Timeline tools
- Cryptographic hash tools

All The Sleuth Kit commands take the file system image as an argument.

```
command [-f fstype] [-i imgtype] [-b dev_sector_size] [-o imgoffset] image
```

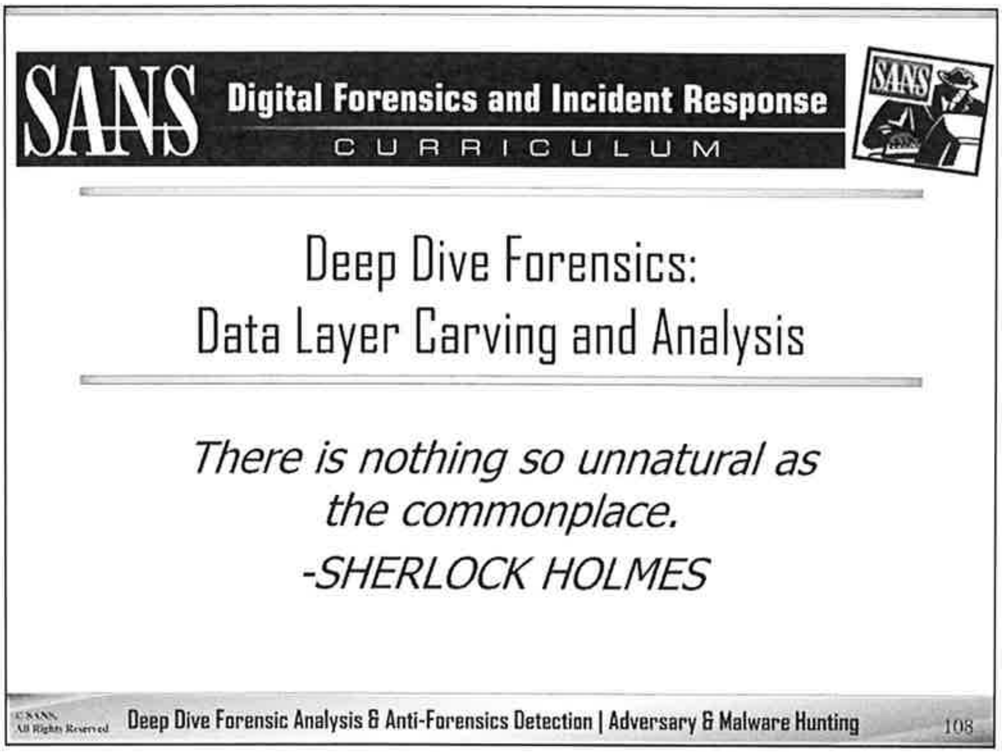
[Useful Options]

`-f fstype:`	Image file system type (use '-f list' for supported types)
`-i imgtype:`	The format of the image file (use '-i list' for supported types)
`-b sector_size:`	The size (in bytes) of the device sectors
`-o imgoffset:`	The offset of the file system in the image(in sectors)

Typically, one also needs to specify the address of "something" (a data unit, metadata structure etc.).

TSK tools can process a partition or disk image file. We will later see that disk images will require additional arguments to specify where the file system starts in the disk.

You can also use split images. Split images are typically created using the 'split' Unix command, which breaks data into equal sized files and the extension increases for each file. To use split images, all files must be specified on the command line or a wild card can be used.



Next, we will look at the tools that analyze the data layer. This is the layer where file data is stored.

Sleuthkit Data Layer Tools

fsstat:	Stats about a given partition
blkstat:	Stats about a given data unit
blkcat:	Display the contents of a given data unit
blkls:	Extracts unallocated data units from an image
blkcalc:	Calculate the original location where a 'blkls' data unit was found
foremost:	Tool that will carve out files from raw images based on the header and footer

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 109

In this next section we will look at multiple tools that focus on the data layer of the filesystem and the partition itself.

The 'fsstat' tool provides general statistics on the file system (hence the 'stat' in the name). The output of this command varies between file system types. As this is a Unix class, we will focus on that output of a Unix image. There are no command line options for 'fsstat' besides the file system type.

The output shows the range of inode values and the range of data fragments. It may also show the last mounted time and the last fsck time (if available).

The partition is divided into groups in a Unix file system and each is grouped in the output. Each group has its own inodes, fragments and copy of the super block. All that information is given in the output. Much of this info is not needed for standard investigations, but can be useful for advanced analysis.

Here we see the first slide of the output from running 'fsstat' on a FAT file system image. The output will take several slides to fully discuss.

The first part of the output gives general identifying information. We also see the number of sectors that are reported to be before this partition. That corresponds to where this partition is located in the disk.

Now we will look at an NTFS image and the fsstat output.

```
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: NTFS
Volume Serial Number: 1884772784770712
OEM Name: NTFS
Volume Name: NTFS-2k-1k
Version: Windows 2000
```

The first part shows serial number and volume name of the image. It also shows it is NTFS from an NT, 2000, or XP system.

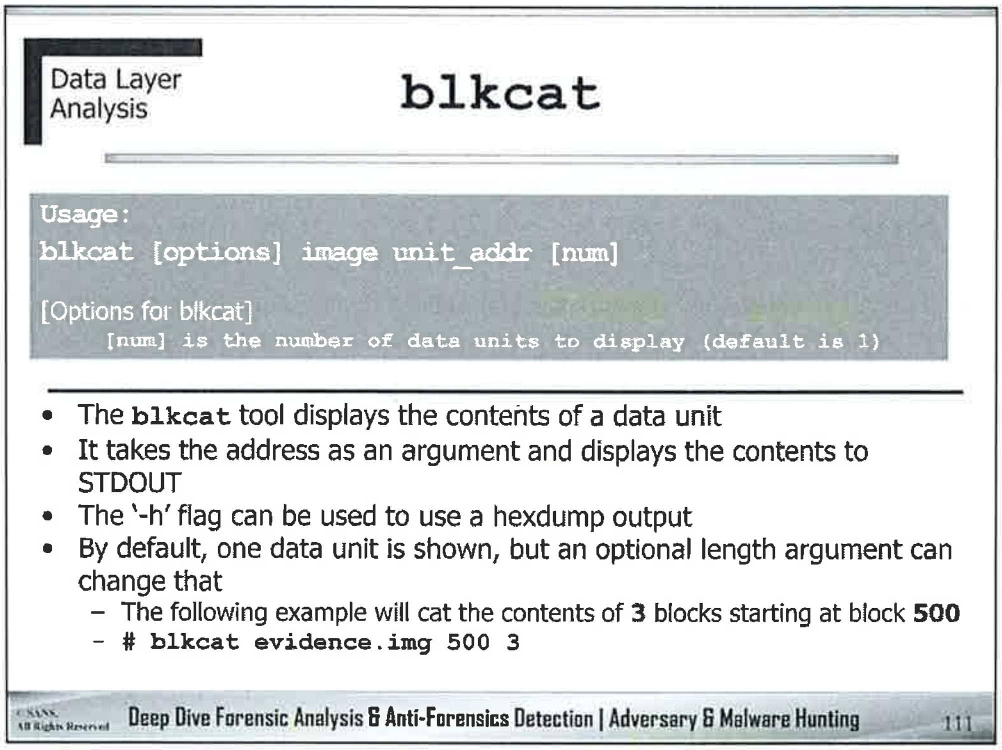
```
METADATA INFORMATION
--------------------------------------------
First Cluster of MFT: 16
First Cluster of MFT Mirror: 52904
Size of MFT Entries: 1024 bytes
Size of Index Records: 4096 bytes
Range: 0 - 4166
Root Directory: 5
```

The second part shows the metadata information. The Master File Table (MFT) is where all of the metadata is stored and its starting address is given, which is cluster 16 in this case. The backup mirror copy address is also given. The sizes of many data structures are dynamic in NTFS and the sizes are given, although you may never need to know these values. The range of metadata values is given as well as the root directory location.

```
CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 1024
Total Cluster Range: 0 - 105807
Total Sector Range: 0 - 211615

$AttrDef Attribute Values:
$STANDARD_INFORMATION (16)   Size: 48-72
   Flags: Resident
$ATTRIBUTE_LIST (32)   Size: No Limit
   Flags: Non-resident
```

The third part of NTFS fsstat output shows the "CONTENT INFORMATION", which describes the cluster and sector sizes and ranges. In this case, the file system image has 1024-bytes clusters and the file system is relatively small with only 105,807 clusters.

The final part of the output is the most obscure. NTFS files are described by their attributes and an NTFS file system can define various aspects of the attributes. This section describes the name of each attribute, its type identifier, its size range, and whether it is stored in the Master File Table (MFT) or in external clusters. For example, the first entry is for the "$STANDARD_INFORMATION" attribute, which has a type of 16, a size range of 48 to 72 bytes, and it is always located in the MFT.



Usage:
`blkcat [options] image unit_addr [num]`

[Options for blkcat]

`[num] is the number of data units to display (default is 1)`

The blkcat tool displays the contents of any data unit. The address is given as an argument and its contents are shown. It can also display the data in a hexdump fashion, which is useful for reverse engineering data.

The same functionality of this tool can be had with the 'dd' tool, but this makes it a little bit easier. The syntax for the 'blkcat' tool is the image name and then the address. If the number of bytes to display is given, it goes after the address:

The blkstat tool is fairly boring. It takes the address of a data unit as an argument and displays statistics about it. This is most useful for identifying if the data unit is allocated or not. 'blkstat' does not have any other arguments besides the file system type.

Usage:
`blkstat [options] image`

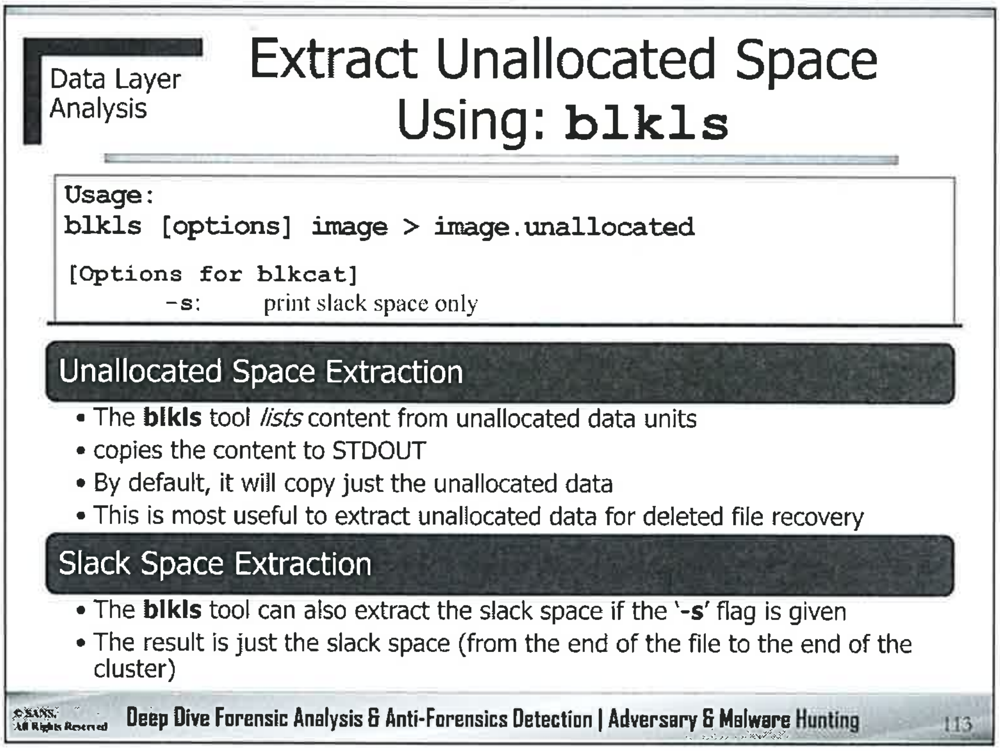
[Default Options Across All Sleuthkit Tools]

`-f fstype:`	Image file system type (use '-f list' for supported types)
`-i imgtype:`	The format of the image file (use '-i list' for supported types)
`-b sector_size:`	The size (in bytes) of the device sectors
`-o imgoffset:`	The offset of the file system in the image(in sectors) from `mmls`

The **blkstat** tool provides *statistics* on a given data unit.
Typically this just indicates its allocation status and in a Unix file system the group that it is a part of

EXAMPLE ONLY:

```
# blkstat evidence.img 368055
Fragment: 368055
Not Allocated
Group: 11
```



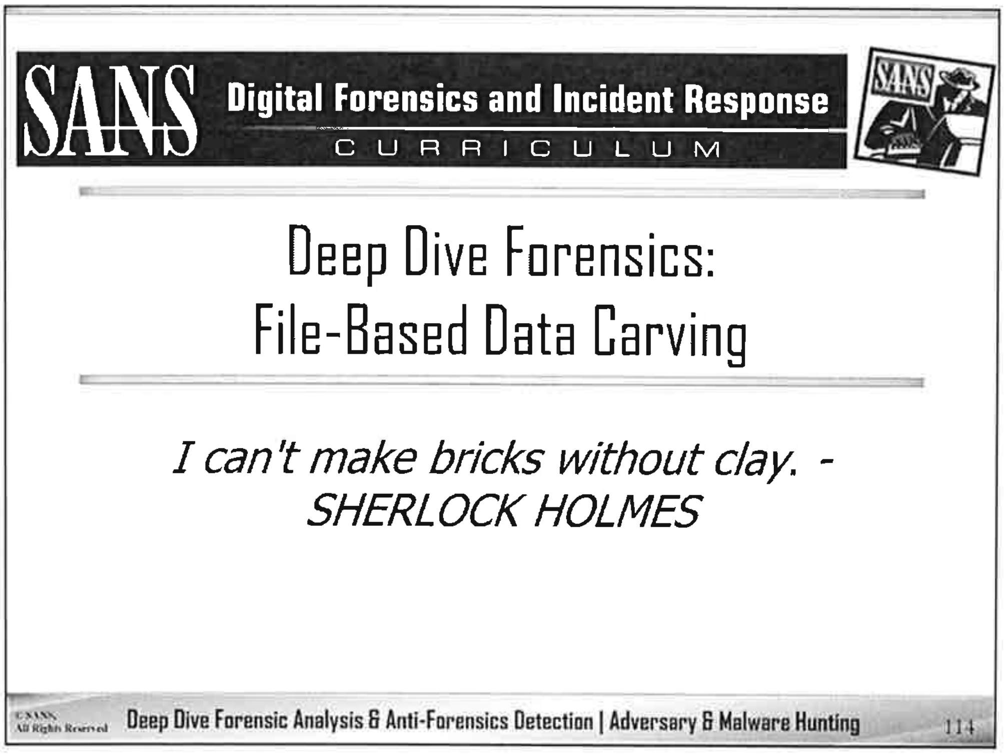
The 'blkls' tool outputs data to stdout. Its purpose is to extract the unallocated data so that deleted data could be recovered.

The result is just a collection of random data. It cannot be mounted or processed with any intelligent tools besides grep and strings.

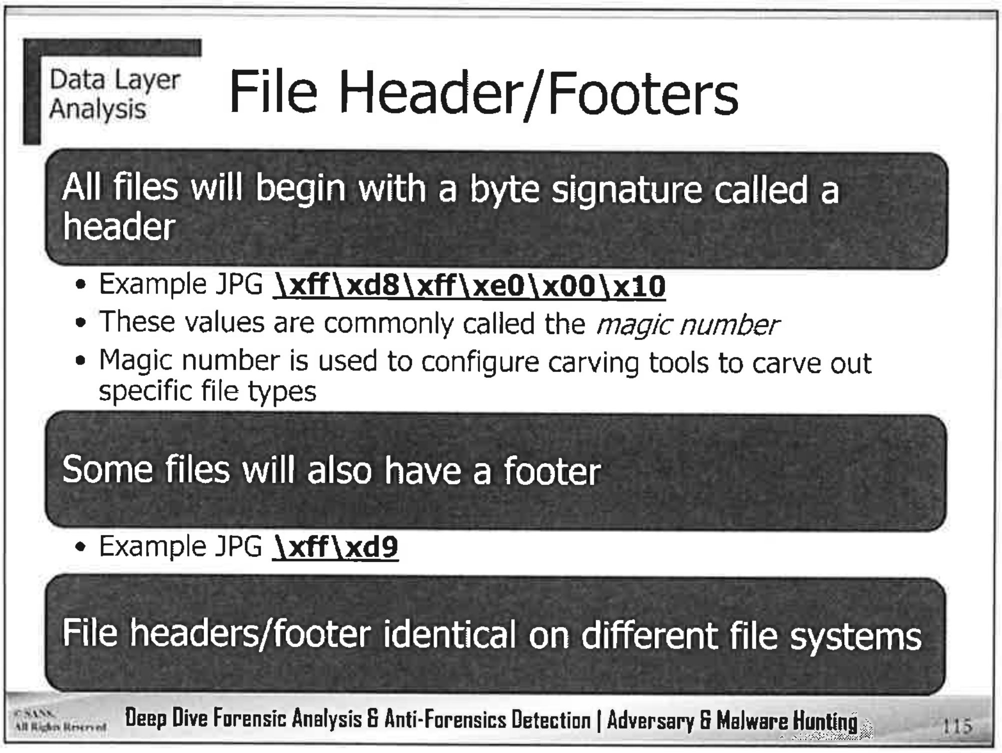
blkls is essentially an "electronic dumpster diving tool". That is a very accurate (and visual) description. blkls scans the file system, collecting bit from here and pieces from there. It focuses on collecting unallocated portions of the file system. As a result, the data collected by this command can use up a lot of space.

The **blkls** tool can also be used to extract the slack space for analysis (keyword searching). The '-s' flag will tell 'blkls' to look at each file and print the data from the end of the file to the end of the cluster. The result is a file of raw data with no structure.

'blkls' extracts the final data unit of each file and zeros out the data that are actual file content. Only the slack space will be left. The output of 'blkls -s' will be a multiple of the data unit size.



Next, we will look at the tools that analyze the data layer. This is the layer where file data is stored.



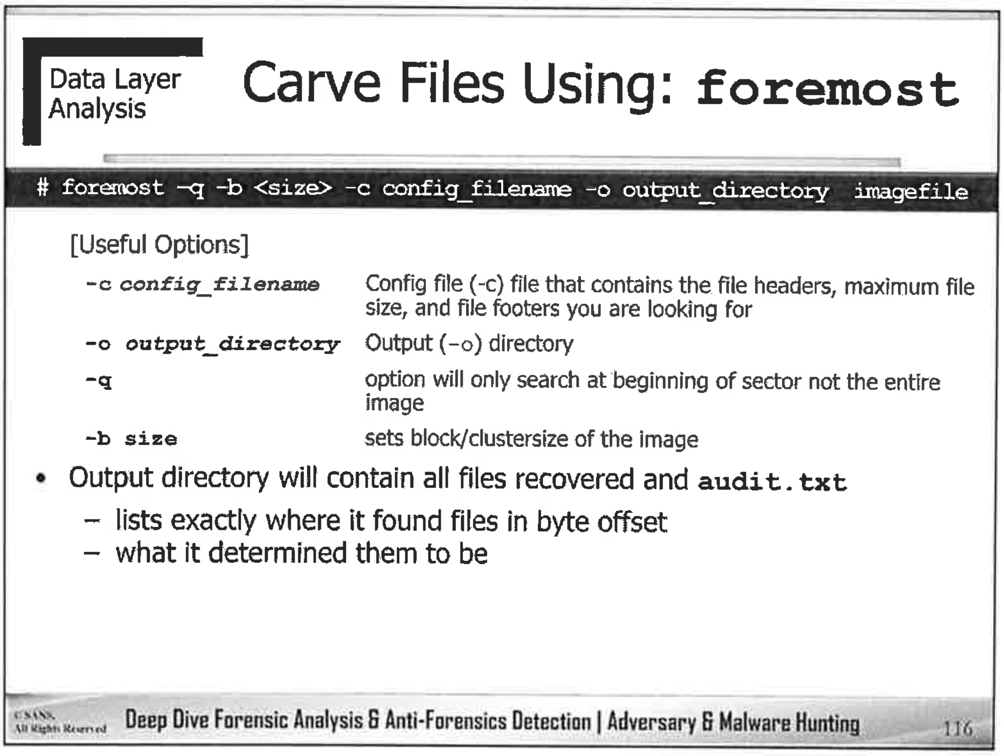
Every file will begin with a byte signature called a header. The header is a key component to computer forensics as it is easily searched for in binary data. Every file will have a specific header that can be searched for regardless which file system it is found on. From an XBOX360, to a camera card, to a USB key, to Server 2008 server a jpeg picture will be the same.

The saying "If you know what you are looking for, and if it exists, you will find it." is true because, if you are looking for a specific byte sequence, you will find the file in question or it probably did not exist.

File Command Overview:

- Can classify unknown files into various types
- Handy for identifying archives, shell scripts, and binaries from large groups of files
- Performs 3 sets of tests:
  - File system tests
  - Magic number tests
  - Language tests
- First successful test causes type to be printed
- `file [options] filename`
  - *filename* can be wildcard (i.e. '*') or list of filenames

The tool file will examine the contents of the data segment of a file. The file tool will perform several tests to categorize the data of a file. It usually does this by looking for the "magic" number. This is a file signature at the beginning of the file that is identical in every file of that file type. (e.g. JPEGS, AVI, MPEG, BMP, etc.) Every file has a unique file signature to it.



Written by Air Force Office of Special Investigation Agents Jesse Kornblum and Kris Kendall Foremost is one of the best tools in an investigator's bag of tricks. Foremost, like Lazarus, is an automated tool to carve out recognizable data from any binary source. This tool, when used on an image, swap space, or as you will learn later, unallocated space, will be critical to helping you recover files that ordinarily have been fairly difficult to recover since the inode data may have been corrupted.

Data Layer Analysis

foremost.conf

- **Edit** `/usr/local/etc/foremost.conf`
- **Uncomment any line that you would like to search for a specific file**

```
#     extension    case        size       header                    footer
#                  sensitive
#
#-----------------------------------------------------------------------------
# GRAPHICS FILES
#-----------------------------------------------------------------------------
#
# AOL ART files
#     art          y           150000     \x4a\x47\x04\x0e          \xcf\xc7\xcb
#     art          y           150000     \x4a\x47\x03\x0e          \xd0\xcb\x00\x00
#
# GIF and JPG files (very common)
      gif          y           155000     \x47\x49\x46\x38\x37\x61 \x00\x3b
      gif          y           155000     \x47\x49\x46\x38\x39\x61 \x00\x00\x3b
      jpg          y           200000     \xff\xd8\xff\xe0\x00\x10 \xff\xd9
#
```


The foremost.conf file is the main file where it will perform searches based on the hex headers and footers of whatever you tell it to look for. The configuration file is populated with a smidgen of the possible types of that could be found at www.wotsit.org.

Headers and footers are decoded before use. To specify a value in hexadecimal use \x[0-f][0-f]. Spaces can be represented by \s.

To specify a value in hexadecimal use \x[0-f][0-f], and for octal use \[0-3][0-7][0-7]. Spaces can be represented by \s. Example: "**\x4d\x5a\x90\x00**" decodes to "**MZ\x90\x00**".

To match any single character (aka a wildcard) use a '?' such as MZ?\x00.

For each file type the configuration file describes the file's extension whether the header and footer are case sensitive the maximum file size and the header and footer for the file. The footer field is optional but header size case sensitivity and extension are not!

Any line that begins with a '#' is considered a comment and ignored. Thus to skip a file type just put a '#' at the beginning of that line.

To match any single character (aka a wildcard) use a '?'. If you need to search for the '?' character you will need to change the 'wildcard' line *and* every occurrence of the old wildcard character in the configuration file.

The NEXT keyword after a footer instructs foremost to search forwards for data that starts with the header provided and terminates or is followed by data in the footer -- the footer data is not included in the output. The data in the footer when used with the NEXT keyword effectively allows you to search for data that you know for sure should not be in the output file. This method, for example, lets you search for two 'starting' headers in a document that doesn't have a good ending footer and you can't say exactly what the footer is but you know if you see another header that should end the search and an output file should be written.

Foremost Configuration File

```
Hex
      \x4d\x5a\x90\x00
ASCII
      MZ\x90\x00
Wildcards = ?
      MZ?\x00
```

EXAMPLES

```
shell_scr y 20000       \x23\x21\x2f\x62\x69\x6e\x2f\x73\x68
\x00\x00

elf y 20000 \x7fELF
gzip y 1500000 \x1f\x8b\x08
```

More Exact Executable Carving: `pe_carve.py`

```
Usage:
pe_carve.py image.blkls
```

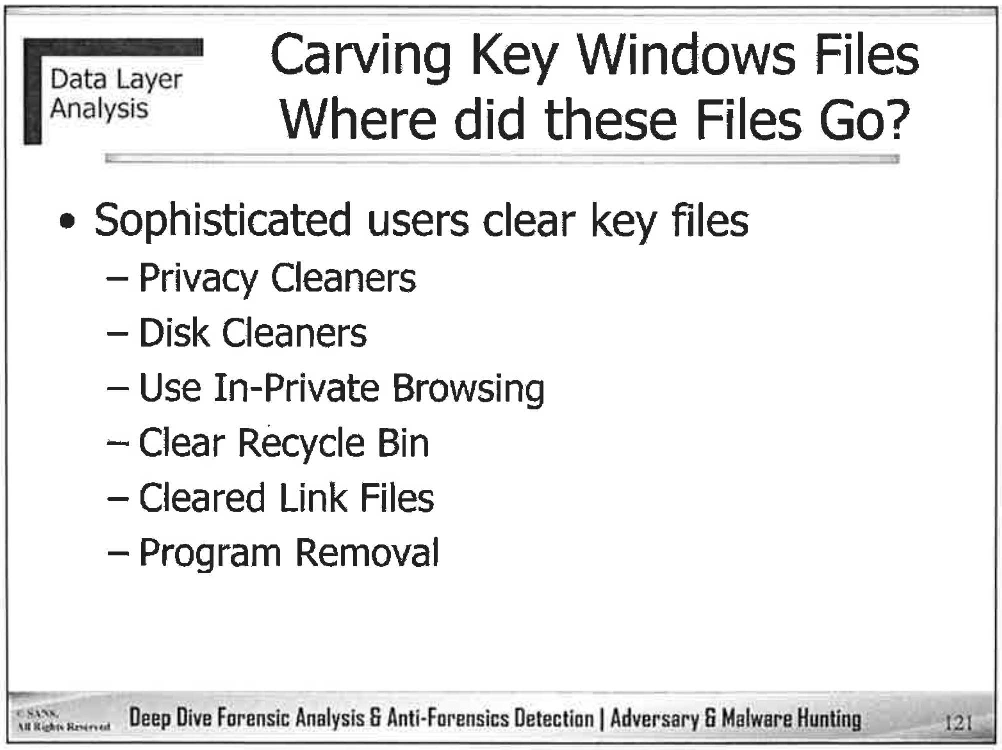
- Windows executables have the file size in the header of the executable
- **`pe_carve.py`** simply scans unallocated space looking for the text "This program" found in nearly all executables.
- It attempts to read the exe header, finds the file size, and extracts out the number of bytes to save
- **`pe_carve.log`** lists all files found and their offset/size
- **`*.livebin`** – extension for recovered files using **`pe_carve.py`**

Windows executables have the file size in the header of the executable. `pe_carve.py` simply scans unallocated space looking for the text "This program" found in nearly all executables. It attempts to read the exe header, finds the file size, and extracts out the number of bytes to save.

pe_carve.py in use

```
root@siftworkstation:/cases/xp-tdungan-c-drive# mkdir pe_carve
root@siftworkstation:/cases/xp-tdungan-c-drive# cd pe_carve/
root@siftworkstation:/cases/xp-tdungan-c-drive/pe_carve# ln -s ../xp-tdungan-c-drive.blkls xp-tdungan-c-drive.blk
ls
root@siftworkstation:/cases/xp-tdungan-c-drive/pe_carve# pe_carve.py xp-tdungan-c-drive.blkls
Scan started on xp-tdungan-c-drive.blkls at [27 Oct 14 @ 22:50:29]
Gathering search hits...
Found at: 0xE6A1D000 (23040 bytes)
Found at: 0xE6A23000 (49664 bytes)
Found at: 0xE6A30000 (985088 bytes)
Found at: 0xE6B21000 (53760 bytes)
Found at: 0xE6B2F000 (177664 bytes)
Found at: 0xE6B5B000 (82432 bytes)
Found at: 0xE6B70000 (19968 bytes)
Ignored PE header at 0xE6BB9000
Ignored PE header at 0xE6BBB000
Ignored PE header at 0xE6BBF000
Ignored PE header at 0xE6BC2000
Ignored PE header at 0xE6BC5000
Ignored PE header at 0xE6BCC000
Ignored PE header at 0xE6BDA000
Ignored PE header at 0xE6BDF000
Ignored PE header at 0xE6BE4000
Ignored PE header at 0xE6BE9000
Ignored PE header at 0xE6BEE000
Ignored PE header at 0xE6BF3000
Ignored PE header at 0xE6BF8000
Ignored PE header at 0xE6BFD000
Ignored PE header at 0xE6C03000
Ignored PE header at 0xE6C13000
Found at: 0xE6CFC000 (997376 bytes)
Found at: 0xE6DF0000 (249856 bytes)
```


This is an example of pe_carve.py in use.

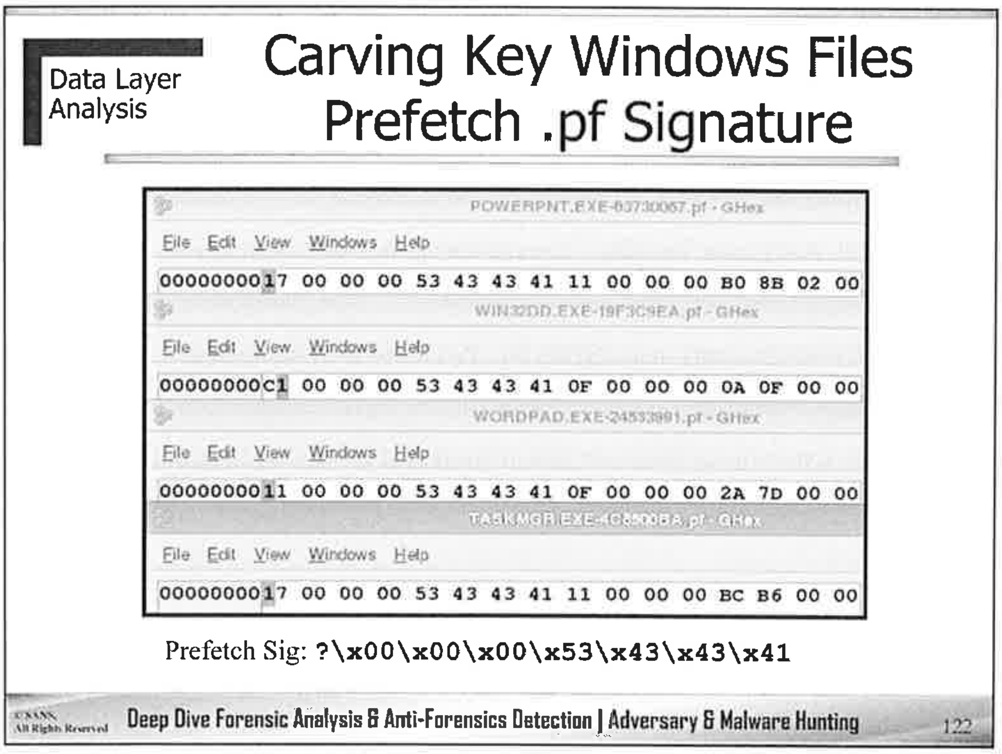


There are many things that could end up deleting key forensic files. Most users know how to clear their history or might even opt to use IE8 In-Private browsing mode. If a privacy cleaner is utilized, many places around the filesystem, including the registry would be affected. Theses cleaners are pretty easy to track by examining many of the same locations on the system for their use including userassist, prefetch, and last commands executed.

The following type of activities could result in files being wiped or deleted from the system. Usually the files are merely deleted.

Privacy Cleaners
Disk Cleaners
Use In-Private Browsing
Clear Recycle Bin
Cleared Link Files
Program Removal

The main idea here is that most of the above listed methods are recoverable via using string and byte searching through the system. How do we know what to look for? We have to examine the artifacts and strings that reside in these critical files.



This is an example of performing analysis of a key file to recover it from unallocated space if it was cleared. I have pulled multiple files from XP, VISTA, and Win7 machines to compare the .pf file on each. As you can see there is a similarity between all of them. In this case the first byte looks like it will need to be a wildcard. Since this is at a sector boundary it makes since to search for the first byte as well using the foremost signature marker for a wildcard, the ?.

Here is the INFO2 file viewed through WINHEX. Notice the occurrences of both UNICODE and ASCII strings in the file. This may be a good way to find a lost INFO2 file.

A way to do this is to find the file on the system using Autopsy and see its original location, perhaps as shown in C:\Documents and Settings\Administrator\Desktop\zcsetup.exe. Perform a search for the hex values that would make up that ASCII word. One of the times you hit, it could be your INFO2 file.

This is where your skills really come into play. You can use one method to lead another location to find out more information.

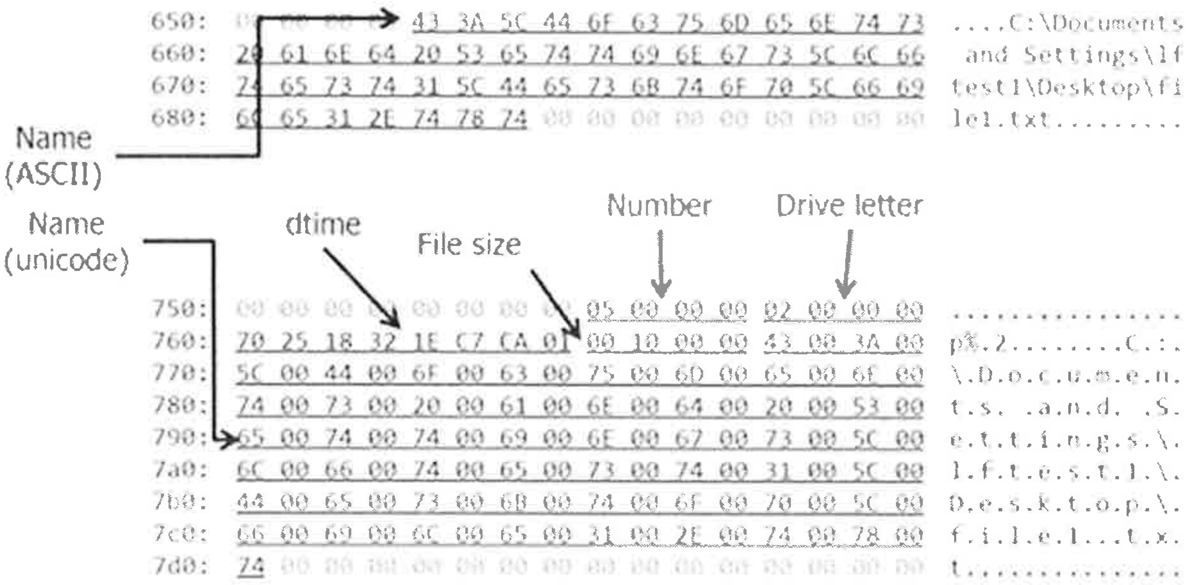
This slide shows the layout of an entry in an INFO2 file. Part of the entry has been removed for readability purposes.

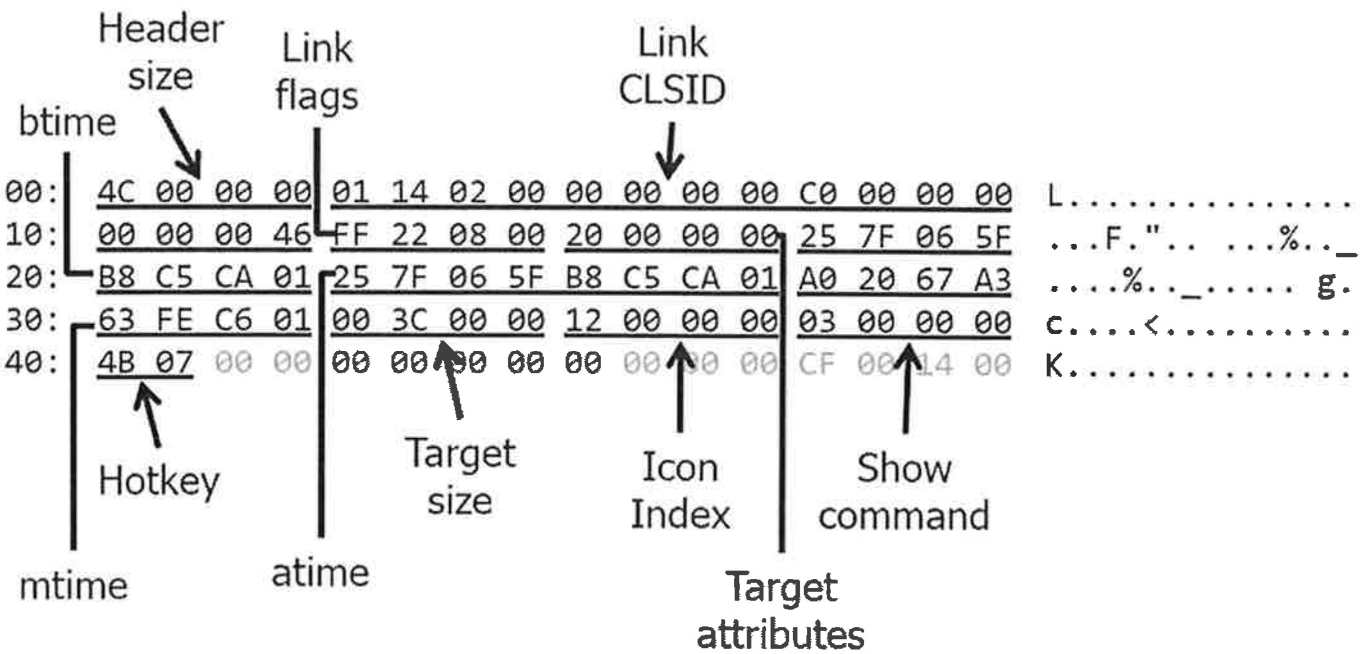
At the start of the entry is the original name and path of the deleted file as a NULL terminated ASCII string, up to a maximum of 260 characters. If the value in this field is less than 260 characters, it is padded with NULL bytes.

The next field is four bytes long and describes the deletion entry number. This is the same number that that you find in the name of the deleted file. The next four bytes describe the drive letter the file was deleted from. A = 0, B = 1, C = 2, and so on.

The next 8 bytes are the time the file was deleted in FILETIME format. Following this field are four bytes denoting the size of the deleted file. Finally is a NULL terminated, 16-bit little endian unicode, version of the file name. This field is 520 bytes long, and is padded with NULLs if the unicode version of the name is less than 520 bytes.

Incidentally, if the first byte of the ASCII version of the file name is NULL (0x00), then it means the corresponding file does not exist in the recycle bin directory.





This picture shows the layout of a ShellLinkHeader. This is the header at the start of an lnk file.

The first four bytes describe the size of the header, and per Microsoft must always be the value 0x4C. The next 16 bytes are a CLSID (Microsoft's term for a GUID) for lnk files. The next four bytes are flags describing the various sections that exist in the lnk file.

The next field is a 4-byte field that describes the attributes of the target, followed by the creation time of the target as an 8-byte FILETIME timestamp. The next two fields are 8 bytes each, and are the access and modification timestamps for the target (again in FILETIME format).

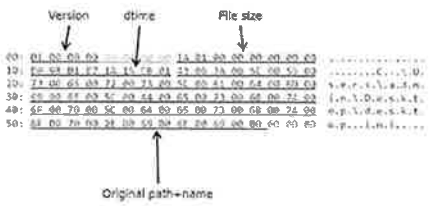
After the modification time, is the size of the target as a 4-byte field, followed by a 4-byte field describing the location for an icon associated with the lnk file. The show command is a 4-byte field that describes how a console window should be displayed (if a console window is used). The last two bytes of the header describe the hotkey combination used to activate the shortcut.

Data Layer Analysis

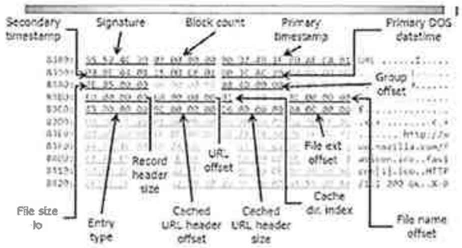
Dissecting Key Windows Files

- Reference USB /Documents:
  - Windows Artifacts Dissected by Mike Murr
- Great reference for internals to key windows files

Recycler: $I<hash> Layout



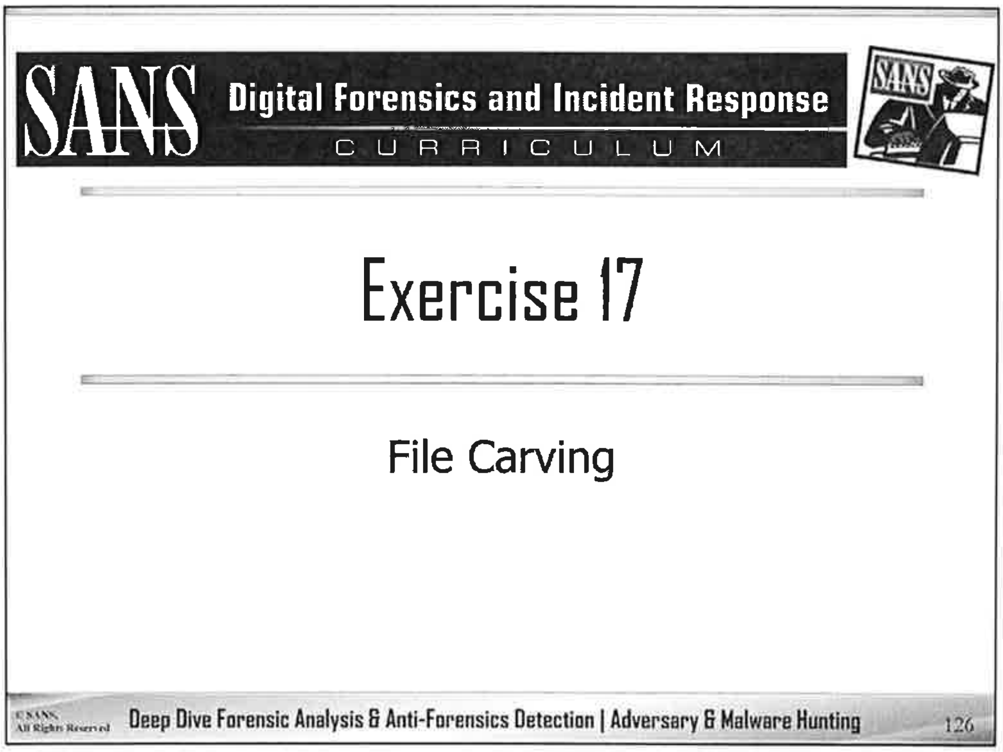
Index.dat: URL Entry (1)



Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 125

Covered in this material to be reviewed and used as a reference on your cases out of class.

Registry Key and Cell Layout
Shell LNK Files Format
INDEX.DAT File Format
INFO2 and $I File Formats
Prefetch File Formats
Office OLE Structure Formats



This page intentionally left blank.

Data Layer Analysis

Data Layer Analysis Review

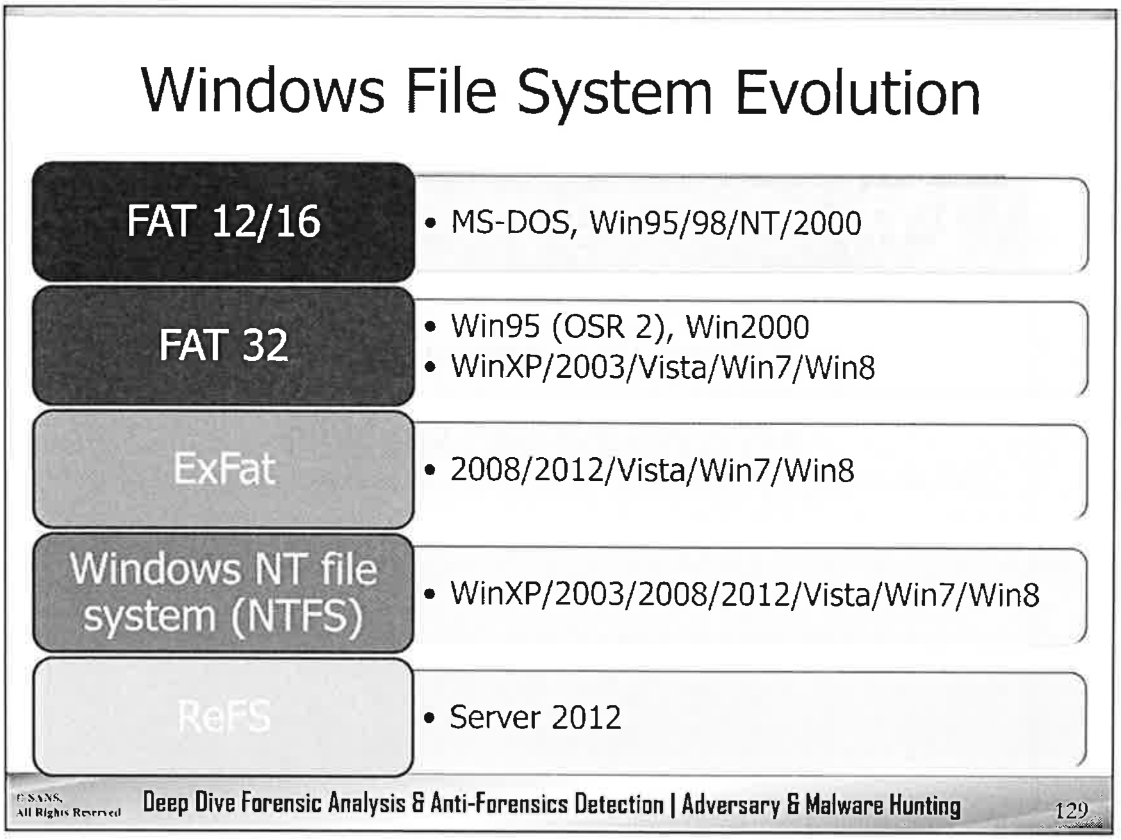
fsstat:	Stats about a given partition
blkstat:	Stats about a given data unit
blkcat:	Display the contents of a given data unit
blkls:	Extracts unallocated data units from an image
blkcalc:	Calculate the original location where a 'blkls' data unit was found
foremost:	Tool that will carve out files from raw images based on the header and footer

This page intentionally left blank.

SANS Digital Forensics and Incident Response
CURRICULUM

Deep Dive Forensics: NTFS Filesystem Analysis

This page intentionally left blank.



A file allocation table (FAT) is a table that Windows maintains on a hard disk that provides a map of the clusters that a file has been stored in.

The number at the end of the FAT is a multiple of how many clusters can be addressed on the file system. For example, on a FAT16 system, it can address 2 to the 16th power or 65,536 clusters.

Windows creates a FAT entry for a new file that records where each cluster is located and its sequential order. When you read a file, the operating system reassembles the file from clusters and places it as an entire file where you want to read it. For example, if there is a long web page, it may very well be stored on more than one cluster on your hard disk.

With 32-bit FAT entry (FAT32) support in Windows 95 OSR2, the largest size hard disk that can be supported is two terabytes!

The FAT file system has been around since the early 1980s and is one of the most simple file systems. It contains no security features, few time stamps, and several hacks that have allowed it to still be used today. There are four variations of FAT: FAT12, FAT16, FAT32, exFAT. The major difference in each is the size of addressable entries in the FAT, which will be described later. The exFAT file system is the newest version and can be found in Windows versions after VISTA SP1 and latest versions of Windows CE 6.0.

FAT32 was introduced with Windows 95 OSR2 or later.

The FAT file system is one the most common PC file system around as it is compatible with so many different computers. FAT is very reliable since it keeps a table of files and free space.

The extended FAT file system (exFAT) was introduced with Windows CE 6.0 in September 2006. It was provided for desktops and server support as SP1 for Vista and Server 2008 in March 2008. Windows XP support was provided in January 2009 in KB Q955704. The file allocation table (FAT) behaves differently than earlier FAT file systems.

NTFS Overview

- A robust file system designed for WinNT
- Security and other features built in
- 4KB Clusters = less slack
- Multiple Versions: 1.0, 1.1, 1.2, 3.0, 3.1

Limits	Theory	Reality
Max File Name Length		255 characters for name 32,767 characters for path
Max File Size	16EB-1KB	16TB-64K
Max Volume Size	$(2^{64}-1)$ * cluster size	256TB (2TB limit imposed by MBR partition, use GPT to get full size)
Max Number of Files		2^{32}-1 (4,294,967,295)

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 130

In 1989, Microsoft and IBM partnered on a new operating system called OS/2 that would be more feature-rich than DOS. This OS failed to catch on, but several of the ground-breaking technologies developed for it were later incorporated into Windows NT. One of those features was OS/2's file system, High Performance File System (HPFS). NTFS was specifically designed to overcome all of the shortcomings of FAT and be applicable in an enterprise environment. Therefore, it included features such as:

- support for mixed case file names, in different code pages
- support for long file names (255 characters as opposed to FAT's 8+3 characters)
- less fragmentation of data
- extent-based space allocation
- transaction journaling for crash recovery
- a B+ tree structure for directories
- support for compression, encryption, quota enforcement at the file system layer
- support for sparse files, soft links, hard links
- POSIX support

The default cluster size for FAT could quickly grow to 64KB, which led to a lot of wasted space as slack behind small files. Therefore, NTFS was designed to keep the cluster size at 4KB for as large a volume as possible. The format command will default to 4KB clusters, but the default can be overridden to larger sizes if the user wishes. [1]

Volume size	Cluster size
7 MB – 512 MB	4 KB
512 MB – 1 GB	4 KB
1 GB – 2 GB	4 KB
2 GB – 2 TB	4 KB
2 TB – 16 TB	4 KB
16TB – 32 TB	8 KB
32TB – 64 TB	16 KB
64TB – 128 TB	32 KB
128TB – 256 TB	64 KB

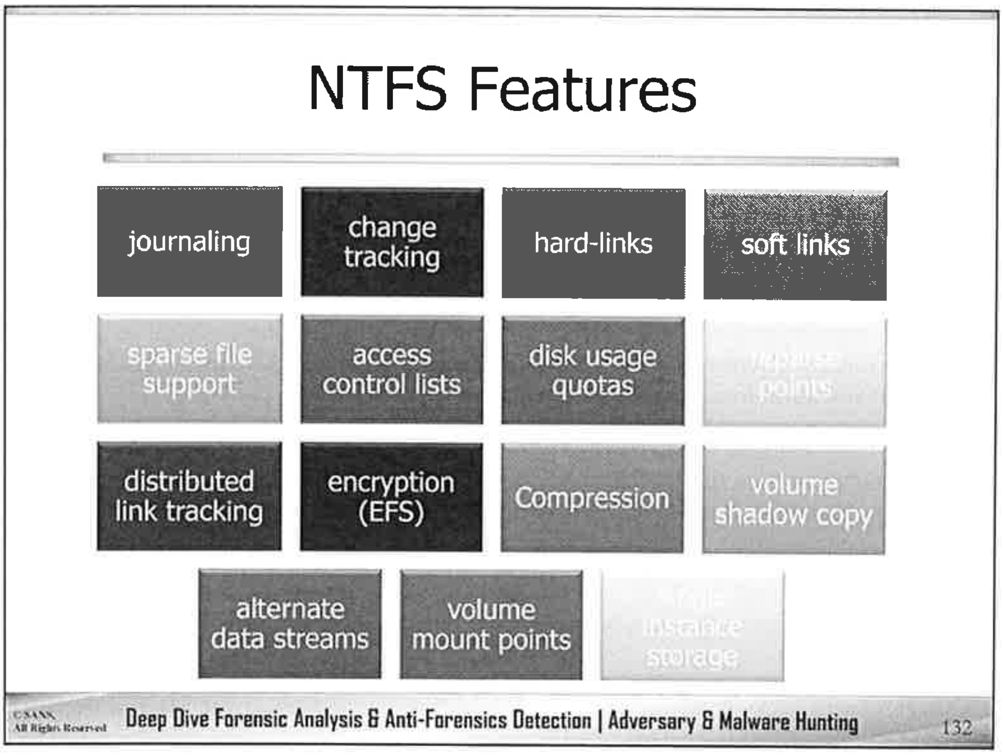
[1] http://support.microsoft.com/kb/140365

Versions:
v1.0 released with NT 3.1 in mid-1993
v1.1 released with NT 3.5 in fall 1994
v1.2 released with NT 3.51 in mid-1995
v3.0 released with Windows in 2000
v3.1 released with Windows XP in autumn 2001

The most significant change was with version 1.2. The next most significant changes to the on-disk structure of NTFS occurred with the release of v3.1, when an extra field was added to the MFT record to explicitly state what record number it was. More on both of these changes in a little bit.

There is a difference between the on-disk format version and a version of the NTFS.sys driver that Windows uses to load the file system. Unfortunately, a lot of marketing material, books, white papers, and so on don't differentiate to which they are referring, leading to references to NTFS v4, v5, and v5.1, which are actually references to the driver version number and the new features that it supports, not to new changes to the on-disk format.

For the maximum file size and max volume size, there is large disparity between what the file system could handle based on the size of the numbers tracking the various parts and what Microsoft actually implemented as the maximum values. For the max file size, the number that tracks the size of the file is a 64-bit number – 2^{64} = 16 exabytes, but within the drivers there is a cap at 16 terabytes (2^{44} – 64KB). Similarly, with the volume size, the number that tracks the clusters is a 64-bit number, but the drivers track that number as a 32-bit number, limiting volume size to 2^{32} minus 1 clusters multiplied by the cluster size.



Wow! NTFS has a LOT of features. The important take away here is that NTFS was designed to be a robust and feature-full file system that can do a LOT more than FAT. Many of these features are only applicable in an enterprise environment, and aren't even fully in use there.

- NTFS makes use of a log file to track changes to the metadata in order to track the state and integrity of the file system at all times and in order to correct inconsistencies causes by system crashes. Other file systems call this "journaling", while NTFS calls it "transaction logging".

- NTFS is able to track all the files that have changed on the system via a USN Journal or Change Journal. This allows programs like a backup utility or virus scanner to know what files are new or changed since they last ran when they need to do an incremental pass over the drive.

- POSIX compliance requires NTFS to support hard links and soft links. A hard link is when a single file responds to multiple names – you at the user level see two files, but you are really only interacting with one. A soft link is where a second file is created, but doesn't have any data itself. Instead it is just an alias or pointer to another file, and opening it actually opens the other file's data without telling you.

- NTFS has incredibly robust security controls to prevent users from opening files they aren't allowed to. (of course, that can all be bypassed by mounting the drive in Linux, but that is a whole other discussion).

- NTFS allows administrators to limit users to a specific amount of disk space they are allowed to consume. No matter which folder they write to, the file system will track all the files that they own and will stop them if they are using too much space.

- Reparse points allow the system to interact with files is all kinds of exciting ways. Soft links, volume mount points, and single instance storage are all implemented via reparse points, just to name a few. And, anyone can program a file system filter driver and create their own reparse points that do whatever they want them to do.

- NTFS uses Object IDs to track certain files, so that no matter where you move a file or how many times you rename it, the distributed link tracking system will update all the links to the file so that you never lose it.

- NTFS implemented file-level encryption at the file system level so that it operates completely transparent to you as the user, but makes it very difficult for others to read your files.

- NTFS implemented file-level compression at the file system level so that it operates completely transparent to you as the user, but saves you considerable disk space.

- NTFS keeps backups of your files as you modify them via the Volume Shadow Copy feature. Did you just delete large parts of a file and hit 'Save' and then realize that you meant to hit 'Save As'? No problem, just revert to a previous version of the file that was saved by NTFS without you even knowing it was done.

- NTFS allows files to have alternate content! For instance, files downloaded from the Internet to be tagged so that Windows can warn you before executing them, for just one of many examples how alternate data streams are used legitimately. Bad guys also use them as a way to hide data and be sneaky.

- NTFS allows you to mount another drive as a folder on the current drive, so instead of having a C: and D: you can have a C: and a C:\data. Very useful for keeping directory tree organized, but still benefiting from the increased speed and volume size by spreading your data across multiple drives.

- NTFS can save disk space on large servers by only keeping one instance of a file. Say, I send a 300GB video of this class to every student in class today, and you all save it on your profile directory on the corporate file server. Dozens of copies * 300GB = a very full file server! Single Instance Storage allows NTFS to reclaim that disk space by converting all of those copies into a single copy of the actual 300GB data and a bunch of very small soft links that point to that single copy, all under the hood without any of you knowing it was happening.

NTFS Volume Boot Record

x00	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
	Jump Code			OEM Signature								bytes/sector | sec/cluster				
x10	00	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00
					Media Descriptor				sec/track |# heads | hidden sectors							
x20	00	00	00	00	80	00	80	00	55	53	C0	6F	00	00	00	00
									Total number of sectors in volume							
x30	00	00	0C	00	00	00	00	00	35	0D	66	03	00	00	00	00
	Start Cluster of $MFT								Start Cluster of $MFTMirr							
x40	F6	00	00	00	01	00	00	00	50	4F	3E	D1	AC	D2	CA	01
	FILE record size				INDX buffer size				Volume Serial Number							
x50	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 134

Above is the first few lines of an NTFS boot sector. Starting at offset x54 is executable code that is part of the boot process. The outlined sections are the pieces of data that are forensically interesting. At offset x03 is the OEM Signature, which tells us we are dealing with NTFS because it says "NTFS".

The two values at offsets x30 and x38 tell us the starting cluster of the $MFT and the location of the $MFTMirr. The VBR is the only construct in NTFS that has to be in the same location (sector 0 of the volume) on every volume. In the event that there is a bad sector at the beginning of the volume where the MFT would normally reside, the MFT can be located elsewhere. This same scenario on FAT or some other file systems results in a "this volume cannot be formatted" error message. So, to recap: VBR is always sector 0; VBR points to start of $MFT; first entry in MFT tells where to read in rest of $MFT; rest of MFT tells us where to read rest of files.

The next two fields, at offsets x40 and x44, tell us the number of clusters each FILE record within the MFT and INDX buffer within an index will consume. INDX size is easy – 1 = each buffer will be 1 cluster (4K). FILE record size is not so easy – xF6 = -10. Being a negative number changes the meaning from straight cluster count to mean we should do some math. Thus: -10 clusters = 2^(-1*-10) = 1024-bytes. In the wild, I have never seen an MFT with a record size other than 1024 bytes and many tools are hardcoded to read that size, but the sheer existence of this field implies that that could change for some unknown reason in the future.

The last field, at offset x48, is the volume serial number. This is a 64-bit number, but almost every tool that displays the volume serial number, such as "DIR" and "VOL" on the Windows command line, will only read the first 32 bits. It is unclear when the system uses the other 32 bits.

Reference Tables 1.12 and 1.13 at http://technet.microsoft.com/en-us/library/cc976796.aspx

NTFS Volume Boot Record

x00	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
	Jump Code			OEM Signature								bytes/sector		sec/cluster		
x10	00	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00
						Media Descriptor			sec/track		# heads		hidden sectors			
x20	00	00	00	00	80	00	80	00	55	53	C0	6F	00	00	00	00
									Total number of sectors in volume							
x30	00	00	0C	00	00	00	00	00	35	0D	66	03	00	00	00	00
	Start Cluster of $MFT								Start Cluster of $MFTMirr							
x40	F6	00	00	00	01	00	00	00	50	4F	3E	D1	AC	D2	CA	01
	FILE record size				INDX buffer size				Volume Serial Number							
x50	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07

Metadata Layer Analysis

NTFS System Files

MFT Record #	File Name	Description
0	$MFT	Master File Table - A database that tracks every file in volume
1	$MFTMIRR	A backup copy of the first 4 records of the MFT
2	$LOGFILE	Transactional logging file
3	$VOLUME	Contains Volume Name, NTFS version number, dirty flag
4	$ATTRDEF	NTFS Attribute definitions
5	.	Root directory of the disk
6	$BITMAP	Tracks the allocation (in-use vs. free) of each cluster in the volume
7	$BOOT	Boot record of the volume
8	$BADCLUS	Used to mark defective clusters so NTFS will not attempt to use them
9	$SECURE	Tracks security information for files within volume
10	$UPCASE	Table of Unicode uppercase characters used to assist sorting file names
11	$EXTEND	A directory containing: $ObjId, $Quota, $Reparse, $UsnJrnl

The first 24 MFT Entries are reserved for special use by the NTFS volume. The first 12 entries are used by system files that make NTFS work. These files are all named starting with a $ and are hidden from view unless using specialized tools. Just like many of the other slides in this section, you don't need to memorize all of this, but it is here for reference.

$MFT

The first record, record number 0, describes the MFT itself. This record provides us the name, $MFT, and information necessary to find all the other clusters containing the rest of the MFT. The Volume Boot Record (VBR) contains a pointer to the cluster this record will lie in, and the records within the MFT contain the pointers to the clusters for every other object. Unlike FAT, in NTFS the VBR is the only object that is tied to a specific sector on disk and cannot be relocated elsewhere.

$MFTMIRR

The second record contains a backup of the $MFT record above in case the above record cannot be read due to physical damage of the disk. The information in record 0 that the system needs to read in the rest of the $MFT file is all we are really needing backed up, but since we are allocating space on the disk for an entire cluster, an entire cluster of MFT records will get backed up – since the default cluster size is 4K and records are 1K, that usually works out to be the first four records.

$LOGFILE

This file contains the Transactional Logging information used by NTFS to maintain integrity of the file system in the event of a crash. This process is called Journaling in most other file systems that support the feature. We will talk more about this file later.

$VOLUME

This file contains the friendly name of the volume for display in My Computer and other locations, as well as the NTFS version number and a set of flags that tell the system if the volume was unmounted cleanly on last use.

$ATTRDEF

This file defines the NTFS Attributes for the version of NTFS in use on this volume. We will talk more about some of these attributes later. The only thing you really need to know is that the names we use to refer to the attributes come from this file – no, we didn't just make them up.

MFT record number 5 is the root directory. Functionally, it is no different than any other directory except that it is always record number 5 and its name is a single dot (".").

$BITMAP

This file is a long string of binary data, with a bit for each cluster within the volume. For each cluster in the volume, the corresponding bit will be set to either 1 or 0 depending on whether the cluster is allocated to a file or not, respectively.

$BOOT

This file allows the VBR to be accessed via normal file I/O operations.

$BADCLUS

This file provides the file system a way of marking, and thus not using, clusters in which there is physical damage making them unreliable to save data there. The $BadClus file is a sparse file that has a file size equal to the volume size and is filled with all zeros. A sparse file is a file in which clusters that are all zeros don't get written to the disk. Since the entire file is all zeros, no space on disk is allocated for the file. If a cluster is determined to be bad, data will be written in this file at the offset that corresponds to the location of that cluster. This fake "data" isn't actually laid on the disk, but the existence of this "data" causes the $Bitmap file to mark that cluster as in use, thus no other new files will try to use that cluster in the future. In the real world, the hard disk controller will remap sectors that are failing, so this fail-safe rarely will get any use.

$SECURE

This file contains a index that is used to track the security information for the files on the system. Each individual file will contain security information about who owns the file and who is allowed to open it. This index allows the system a central place to hold information about the owners so that information does not have to be repeated in every file. In older versions of NTFS, this record number (9) held the system file named $Quota, which is discussed below.

$UPCASE

This file contains a table of uppercase and lowercase unicode letters for each unicode code page in use for file names within the system. This table is used in sorting the files by name so that "A" and "a" are next each other when sorting alphabetically.

$EXTEND

Even though there are 24 records reserved for system use, when new system files were introduced, rather than place the new system files in those records, a directory entry was placed in record number 11 to hold the new system files and these new system file were placed into normal records for regular use. Since the files below are the written by the format command before user files are written, they will almost always be located in the first four records that are not reserved (record numbers 24-28), but they are not static like the first 12 records, which are always the record numbers indicated above.

$EXTEND\$ObjId

This file contains an index of all the object IDs in use within the volume. Object IDs allow a file to be tracked even if the file gets moved, renamed, or otherwise changed in a way that would cause a pointer like a link file to be able to find the file.

$EXTEND\$Quota

This file contains information about how much allocated space each user is allowed to and has consumed on this volume, in order for a system administrator to technically prevent a user from using too much disk space.

$EXTEND\$Reparse

This file contains an index of all the reparse points on the volume. Reparse points have a multitude of uses, but the most common is for symbolic links, in which a file is really just a pointer to another file and editing this file is really modifying the file it points to. Reparse points are also used for mounting other volumes as a directory on this volume.

$EXTEND\$UsnJrnl

The Update Sequence Number (USN) Journal, or also known as the Change Journal, is an index listing all of the files that have changed on the system and why the change took place. We will talk more about this journal later.

Reference pages 277-278 of your **File System Forensic Analysis** book.

http://support.microsoft.com/kb/103657/en-us

Master File Table

- Database-like and very structured
- Records are 1024-bytes long
- Every object on volume gets an entry
- Saved in the "MFT Zone"

The Master File Table (MFT) is a very structured database that tracks all the objects to be saved on an NTFS volume. Every object gets a FILE record within the MFT. Each FILE record contains a series of Attributes that contain the various data and metadata related to that file. A file gets a record, a directory listing gets a record, even the volume name gets its own record.

Each record is 1024 bytes long. If a file is small enough, the file's contents will be held within the MFT record right next to its metadata. Otherwise, there is a pointer to which clusters contain the file's contents.

The MFT can become quite large. A 1TB drive with over 400K files on it will produce an MFT that is over 485MB. If the MFT becomes fragmented, and the system has to seek all over the drive to get to its various parts, the speed of the whole system will become very noticeably degraded. To prevent this, NTFS drivers will create an "MFT Zone" for the MFT to reside in it. This reserves the first 12.5% of the drive and starts placing user files after this zone so that the MFT has free space for it to grow into. If the rest of the free space in the other 87.5% of the drive becomes full, then the MFT Zone will be cut in half, and again when that other half gets full, and so on until the drive is full. Once the MFT becomes fragmented, it cannot be defragmented by normal means. (Handy tip from a seasoned sysadmin: Quickest and most thorough way to defrag a drive is to copy all the files off, format, then copy them all back. They will come back on one at a time and NTFS will lay them down in order all contiguous.)

http://msdn.microsoft.com/en-us/library/bb470206(v=vs.85).aspx

Metadata Layer Analysis

MFT FILE Record Header

x00	46	49	4C	45	30	00	03	00	66	60	49	01	00	00	00	00
	"FILE" signature				Offset to Fixup		Fixup size		$Logfile Sequence Number							
x10	02	00	01	00	38	00	01	00	88	01	00	00	00	04	00	00
	Sequence #		Hard Link		Offset 1st Attr		Flags		Real Size of Record				Allocated Size of Record			
x20	00	00	00	00	00	00	00	00	06	00	00	00	76	13	00	00
	File Reference to Base Record								Attr Count				Inode # of this record			
x30	09	00	00	00	00	00	00	00	10	00	00	00	60	00	00	00
	Fixup						Padding		Start of First Attribute							
x40	00	00	00	00	00	00	00	00	48	00	00	00	18	00	00	00

Each MFT entry will begin with signature value of "FILE" (x46 x49 x4C x45). If the system has detected an error in the entry, you may see the alternative signature of "BAAD" (x42 x41 x41 x44). Looking for "FILE" at the start of a sector is a good way of locating MFT fragments in unallocated space.

Following the signature field is a two byte value that gives the offset to the Fixup Array relative to the start of the MFT entry and a second two byte value that tells the number of entries in the Fixup Array. The Fixup Array is used for error checking for MFT data structures that span multiple sectors, excluding sectors that contain file data. When these data structures are written to disk, the last two bytes of each sector are copied into a special array and a signature value is written to the last two bytes of each sector. When the data structures are read by the system, the last two bytes of each sector are compared against the signature value and if everything checks out, the signature values are replaced by the values that were previously copied into the array. In this way, the system is able to detect corrupt entries.

The $LogFile Sequence Number (LSN) at offset x08 is part of the journaling file system and is used to determine whether the file system is consistent, or needs to have certain actions redone or undone to achieve consistency. The number written here is the file offset into the $LogFile where the record pertaining to this inode will be located – x01496066 (or 20MB) for the example above.

The Sequence Number at offset x10 is a counter tracking the number of times the MFT record has been re-used. When an MFT record is allocated for the first time, its Sequence Number will be set to one, when the file is deleted, and the MFT record is unallocated, the Sequence Number is incremented again and will only be incremented on subsequent unallocations.

Reference:
pages 353-355 of File System Forensic Analysis
http://msdn.microsoft.com/en-us/library/bb470124(VS.85).aspx
http://msdn.microsoft.com/en-us/library/bb470211(VS.85).aspx

MFT FILE Record Header

x00	46	49	4C	45	30	00	03	00	66	60	49	01	00	00	00	00
	"FILE" signature				Offset to Fixup		Fixup size		$Logfile Sequence Number							
x10	02	00	01	00	38	00	01	00	88	01	00	00	00	04	00	00
	Sequence #		Hard Link		Offset 1st Attr		Flags		Real Size of Record				Allocated Size of Record			
x20	00	00	00	00	00	00	00	00	06	00	00	00	76	13	00	00
	File Reference to Base Record								Attr Count				Inode # of this record			
x30	09	00	00	00	00	00	00	00	10	00	00	00	60	00	00	00
	Fixup						Padding		Start of First Attribute							
x40	00	00	00	00	00	00	00	00	48	00	00	00	18	00	00	00

The Hard Link count at offset x12 refers to the number of $File_Name attributes there are for this file. Multiple file name attributes could be long and short file names for the same name (in which they are in the same directory and only one or the other will be displayed to you) or they could be true hard links where they are different names or in different directories but they are referring to the same record.

The Flags at offset x16 can have the following values:

0x00	0000	not in use
0x01	0001	file in use
0x02	0010	directory that has been deleted
0x03	0011	directory in use

Deleting a file changes the "in use" bit of the flag to 0, but does nothing to clear out the rest of the data, thus many deleted file's metadata is still recoverable as long as the MFT record hasn't been recycled already.

The File Reference to Base Record at offset x20 is only used for Extended Records, which are only used when a record contains so many attributes that it can't fit them all within 1024-bytes. So, for the vast majority of records, this will be all zeros.

The Attribute Count field at offset x28 essentially tell you how many attributes this record will have. Each attribute will have an ID number, starting at zero.

The next field depends on the version of NTFS you are dealing with. With older versions, the very next byte after the Attribute Count was the start of the Fixup. Starting with Windows XP using version 3.1 and onward, the MFT Record Number (or Inode Number) of this record is saved at offset x2C. This field is very useful when finding fragments of the MFT during data recovery. The inclusion of this field changed the header length, which is noticeable in the Offset to Fixup at offset x04. The value of that field changes from a x2A in older versions to a x30 in newer systems, or in ASCII, it changes from a "*" to a "0". This is important to note because some file carving tools will include that byte in their signature of an MFT record, and if they are looking for "FILE*" but not "FILE0", they may not find what you are expecting to find.

The next three two-byte fields are the Fixup code, made up of the Update Sequence Number (USN) and the Update Sequence Array (USA). The Fixup Length at offset x06 will tell you how many two-byte entries make up the USN+USA. At the end of each 512-byte sector in the 1024-byte MFT Record, the system will write the USN (note: unless MFT record length changes, this will always happen twice, thus for MFT records the fixup will always be three entries). When it reads those sectors in, the system will compare the numbers it finds at the end of each sector with the USN in the record header and if all three match, then everything is great. If they don't match, the "FILE" signature is overwritten with "BAAD" and an error message is returned. As for the original contents of the bytes at the end of the sector that were overwritten, that is where the USA comes into play. Each entry in the USA contains the original two bytes that should be in the last two bytes of each sector of the record. So, when reading in the first sector of the record, the system checks the last two bytes against the USN and then replaces those two bytes in memory with the first entry from the USA. Then it repeats that process for the second sector using the second USA entry.

Metadata Layer Analysis

NTFS Attributes

Type	Name	Type	Name
0x10	**$STANDARD_INFORMATION**	**0x90**	**$INDEX_ROOT**
0x20	$ATTRIBUTE_LIST	**0xA0**	**$INDEX_ALLOCATION**
0x30	**$FILE_NAME**	0xB0	$BITMAP
0x40	$OBJECT_ID	0xC0	$REPARSE_POINT
0x50	$SECURITY_DESCRIPTOR	0xD0	$EA_INFORMATION
0x60	$VOLUME_NAME	0xE0	$EA
0x70	$VOLUME_INFORMATION	0xF0	
0x80	**$DATA**	0x100	$LOGGED_UTILITY_STREAM

Above is a list of the attributes as defined in the $AttrDef system file. The ones in bold are the most common ones and the ones that hold the most important information to us.

$STANDARD_INFORMATION – Provides us time stamps, flags, and tracking information
$ATTRIBUTE_LIST – Used when there are so many attributes that they won't all fit in a single MFT Record
$FILE_NAME – Provides us the parent directory, time stamps, size, flags, and file's name
$OBJECT_ID – Provides a GUID used to track the file as it gets moved around or renamed.
$SECURITY_DESCRIPTOR – Contains security information for the file for enforcement of ACLs
$VOLUME_NAME – Only found in $Volume system file; provides volume's name. Yeah, that's it.
$VOLUME_INFORMATION – Only found in $Volume system file; provides NTFS version and flags
$DATA – Contents of the file
$INDEX_ROOT – Contains the header of an index; tells system how to sort entries, size of entries, etc
$INDEX_ALLOCATION – Like $Data, but for Indexes
$BITMAP – Used with indexes to identify which entries are in use and which are available
$REPARSE_POINT – Tells system to read the file differently than normal via a
$EA_INFORMATION & $EA – Legacy; aren't used any more
$LOGGED_UTILITY_STREAM – Like $Data but for other uses, such as logging info or encryption keys

Every attribute has a standard header that starts with a signature number (indicated in the table above) that identifies which type of attribute is about to be read. Also in this header is a series of length fields to tell the system how long this attribute is, which is how it finds the next attribute. After this attribute header is the attribute's content, which is either present in the MFT or the attribute header tells us where on the disk to find that content. In the case of the $Data attribute, this content is the actual content of the file. Other attributes work the exact same way, but their content is structured data that serves a specific purpose.

Metadata Layer Analysis

Typical FILE Record

File	Directory
`$Standard_Information`	`$Standard_Information`
`$File_Name` (Long)	`$File_Name` (Long)
`$File_Name` (Short - sometimes)	`$File_Name` (Short - sometimes)
`$Data`	`$Index_Root`
	`$Index_Allocation` (sometimes)

$Standard_Information and one $File_Name are mandatory for all files and directories.

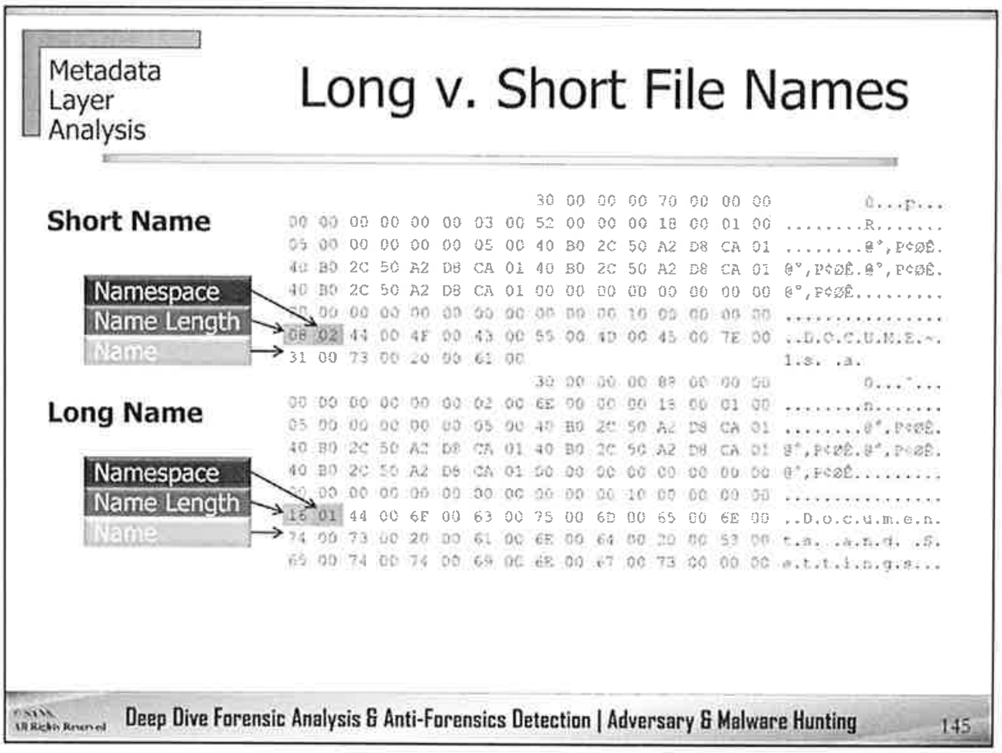
If the name is longer than 8 characters or contains special characters, then Windows will also create a DOS-compatible 8.3 name and save this as a second $File_Name attribute.

Files will have a $Data attribute to hold their content. The $Data attribute can be either resident (completely within the MFT Record) or nonresident (header in MFT that tells you which clusters the content is in).

For directories, their content is an index that lists all of their files. The $Index_Root provides the header for the index. If there are only a few entries and they will fit within the MFT Record, the entries will be part of the $Index_Root. Once the listing is too long to fit within the MFT Record, an $Index_Allocation attribute is used to track which clusters on the disk the index will reside.

The above are the bare minimum to get things to work. Other attributes are possible, but usually only show up if certain criteria are met or features are turned on. Both files and directories can have an $Object_ID and/or a $Security_Decriptor. A file can have a $Logged_Utility_Stream, usually for storing transactional information or encryption keys. And, directories can have a $Bitmap attribute to track which entries in the index are in use and which are free.

In the next few pages we will examine the attributes above in more detail. We have a lot to cover this week, so we are just going to cover the most forensically important attributes.



If a file has a small name, such as "File.txt", only a single $File_Name will be saved. If the file has a long name, such as "Really Long Name.txt", two $File_Name attributes will be created to save both the long name preserving the case and a second one to save a DOS-compatible short name ("REALL~1.TXT"). This gives us potentially 3 different sets of time stamps for the file. As we will see later, entries in directory listings follow the same format as above, giving us yet another set of time stamps for the same file.

Immediately preceding the name are two one-byte fields. The first one tells us the number of two-byte Unicode characters making up the name. This is helpful for us so that we don't read in more than we should as the padding after the name could contain some garbage information, similar to file slack. For example, the "s a" at the end of the top attribute is really just the padding so that the next attribute starts at the correct location.

The next field, is a Namespace Type that tells us the type of name.

x00 – POSIX	Case is preserved and different case is considered a different name
x01 – Win32	A case sensitive long name
x02 – DOS	A DOS-compatible short name
x03 – Win32/DOS	Name is short enough to only need one $File_Name

Long v. Short file names

Short Name

Namespace
Name Length
Name

```
                                    30 00 00 00 70 00 00 00          0...p...
00 00 00 00 00 00 03 00 52 00 00 00 18 00 01 00  ........R.......
05 00 00 00 00 00 05 00 40 B0 2C 50 A2 D8 CA 01  ........@°,PcØÊ.
40 B0 2C 50 A2 D8 CA 01 40 B0 2C 50 A2 D8 CA 01  @°,PcØÊ.@°,PcØÊ.
40 B0 2C 50 A2 D8 CA 01 00 00 00 00 00 00 00 00  @°,PcØÊ.........
00 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00  ................
08 02 44 00 4F 00 43 00 55 00 4D 00 45 00 7E 00  ..D.O.C.U.M.E.~.
31 00 73 00 20 00 61 00                          1.s. .a.
```

Long Name

Namespace
Name Length
Name

```
                                    30 00 00 00 88 00 00 00          0...^...
00 00 00 00 00 00 02 00 6E 00 00 00 18 00 01 00  ........n.......
05 00 00 00 00 00 05 00 40 B0 2C 50 A2 D8 CA 01  ........@°,PcØÊ.
40 B0 2C 50 A2 D8 CA 01 40 B0 2C 50 A2 D8 CA 01  @°,PcØÊ.@°,PcØÊ.
40 B0 2C 50 A2 D8 CA 01 00 00 00 00 00 00 00 00  @°,PcØÊ.........
00 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00  ................
16 01 44 00 6F 00 63 00 75 00 6D 00 65 00 6E 00  ..D.o.c.u.m.e.n.
74 00 73 00 20 00 61 00 6E 00 64 00 20 00 53 00  t.s. .a.n.d. .S.
65 00 74 00 74 00 69 00 6E 00 67 00 73 00 00 00  e.t.t.i.n.g.s...
```

Metadata Layer Analysis

$Standard_Information

x00	10	00	00	00	60	00	00	00	00	00	00	00	00	00	00	00
	Attribute Signature | Length of Attribute															
x10	48	00	00	00	18	00	00	00	D5	49	EB	25	2D	4C	CA	01
									Creation Time							
x20	F5	F1	D6	26	2D	4C	CA	01	20	31	0F	F0	FF	CC	CC	01
	Content Modified Time								Metadata Modified Time							
x30	D5	49	EB	25	2D	4C	CA	01	20	00	00	00	00	00	00	00
	Last Accessed Time								Flags				Max Versions			
x40	00	00	00	00	00	00	00	00	00	00	00	00	AE	03	00	00
	Version Number				Class ID				Owner ID				Security ID			
x50	00	00	00	00	00	00	00	00	68	FE	75	7B	03	00	00	00
	Quota Charge								Update Sequence Number							

The $Standard_Information attribute is identified by a Signature value of x10 or 16.

At offset x18 you will find a set of four time timestamps in the order of Created, Modified, MFT modified, and Last Accessed. These are the four timestamps that are going to be used by the file system to display the time information. Other timestamps will be kept in other locations that we will see later, but these are the definitive set.

At offset x38 will be a set of flags. These flags track the file attributes, such as hidden, read only, and so on.

10 00 00 00 Directory	01 00 00 00 Read Only	00 10 00 00 Offline	00 01 00 00 Temporary
20 00 00 00 Archive	02 00 00 00 Hidden	00 20 00 00 Not Indexed	00 02 00 00 Sparse File
40 00 00 00 Device	04 00 00 00 System	00 40 00 00 Encrypted	00 04 00 00 Reparse Point
80 00 00 00 Normal			00 08 00 00 Compressed

Each value above corresponds to a single bit being set to 1. For a given value, look to see what values adds up to it and that will tell you what flags are set. For example, in the hex dump above, the Flags value is x20 = the Archive bit is set. If it was x26 x08 = the Archive, Hidden, System, and Compressed bits are set.

The rest of the values provide tracking numbers for the file used by the $Secure, $Quota, and $UsnJrnl system files. These will be unique numbers assigned for tracking this specific file in those indexes, and have no meaning outside the file system (the Security ID here has no relation to the SID using in Windows to track the user). They are vitally important when parsing the $Secure and $UsnJrnl as some entries in that index will only refer to the file by the tracking number here. Most of these fields are only present in version 3.0 and 3.1.

Reference pages 360-362 of your **File System Forensic Analysis** book for additional details.

$Standard_Information

x00	10	00	00	00	60	00	00	00	00	00	00	00	00	00	00	00
	Attribute Signature | Length of Attribute															
x10	48	00	00	00	18	00	00	00	D5	49	EB	25	2D	4C	CA	01
									Creation Time							
x20	F5	F1	D6	26	2D	4C	CA	01	20	31	0F	F0	FF	CC	CC	01
	Content Modified Time								Metadata Modified Time							
x30	D5	49	EB	25	2D	4C	CA	01	20	00	00	00	00	00	00	00
	Last Accessed Time								Flags				Max Versions			
x40	00	00	00	00	00	00	00	00	00	00	00	00	AE	03	00	00
	Version Number				Class ID				Owner ID				Security ID			
x50	00	00	00	00	00	00	00	00	68	FE	75	7B	03	00	00	00
	Quota Charge								Update Sequence Number							

Metadata Layer Analysis

$File_Name

x00	30	00	00	00	78	00	00	00	00	00	00	00	00	00	02	00
	Attribute Signature				Attribute Length											
x10	5A	00	00	00	18	00	01	00	05	00	00	00	00	00	05	00
									Parent Directory Reference							
x20	E0	62	1B	4B	A2	D8	CA	01	E0	62	1B	4B	A2	D8	CA	01
	Creation Time								Content Modified Time							
x30	E0	62	1B	4B	A2	D8	CA	01	E0	62	1B	4B	A2	D8	CA	01
	Metadata Modified Time								Last Accessed Time							
x40	00	A0	F6	7F	01	00	00	00	00	00	00	00	00	00	00	00
	Physical File Size								Logical File Size							
x50	26	00	00	00	00	00	00	00	0C	03	68	00	69	00	62	00
	Flags										h		i		b	
x60	65	00	72	00	66	00	69	00	6C	00	2E	00	73	00	79	00
	e		r		f		i		l		.		s		y	
x70	73	00	00	00	00	00	00	00								
	s															

The $File_Name attribute is identified by a Signature value of x30 or 48.

The Parent Directory Reference at offset x18 consists of two parts. The first 6 bytes are the MFT Record number of parent directory, and the remaining two bytes are a sequence number for that MFT Record.

Though the $File_Name attribute contains the same four time stamps as $Standard_Information, these are only updated if this attribute is changed, such as by renaming or moving a file. These time stamps are hard to modify naturally using the win32API. On day 3, we will show you how to use these times to be able to tell when certain types of activity occur in relation to a file. This is very valuable as it ends up being an indicator that file timestamps were backdated using a hacker tool. Many tools simply do not show these timestamps at all. Two tools that do a good job of displaying these time stamps are MFTView [1] from Sanderson Forensics, which easily allows you to parse each individual attribute using a hex-editor-like GUI, or the AnalyzeMFT.py [2] Python script by David Kovar, which is part of the SIFT workstation.

The Flags at offset x50 are the same as the ones in the $Standard_Information attribute.

Each attribute must start at a increment of 8 bytes, so there will be 0-7 bytes of padding after the file name.

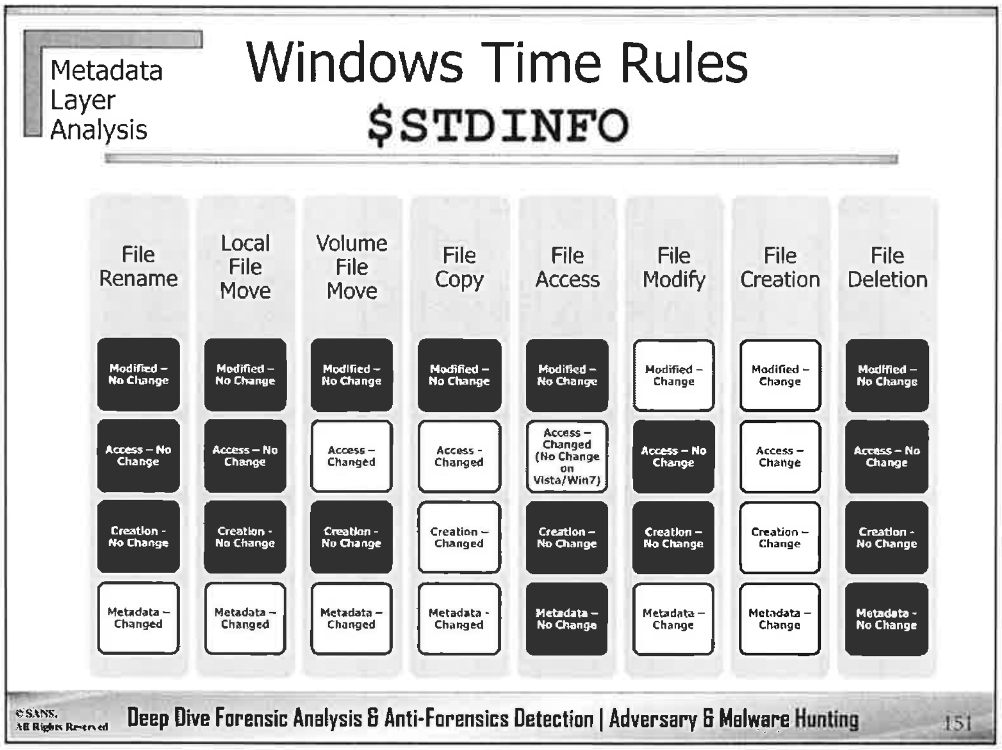
[1] http://www.sandersonforensics.com/forum/content.php?133-MFTView

[2] http://www.integriography.com/

Reference pages 363-364 of your **File System Forensic Analysis** book.

$File_Name

x00	30	00	00	00	78	00	00	00	00	00	00	00	00	00	02	00
	Attribute Signature				Attribute Length											
x10	5A	00	00	00	18	00	01	00	05	00	00	00	00	00	05	00
									Parent Directory Reference							
x20	E0	62	1B	4B	A2	D8	CA	01	E0	62	1B	4B	A2	D8	CA	01
	Creation Time								Content Modified Time							
x30	E0	62	1B	4B	A2	D8	CA	01	E0	62	1B	4B	A2	D8	CA	01
	Metadata Modified Time								Last Accessed Time							
x40	00	A0	F6	7F	01	00	00	00	00	00	00	00	00	00	00	00
	Physical File Size								Logical File Size							
x50	26	00	00	00	00	00	00	00	0C	03	68	00	69	00	62	00
	Flags										h		i		b	
x60	65	00	72	00	66	00	69	00	6C	00	2E	00	73	00	79	00
	e		r		f		i		l		.		s		y	
x70	73	00	00	00	00	00	00	00								
	s															

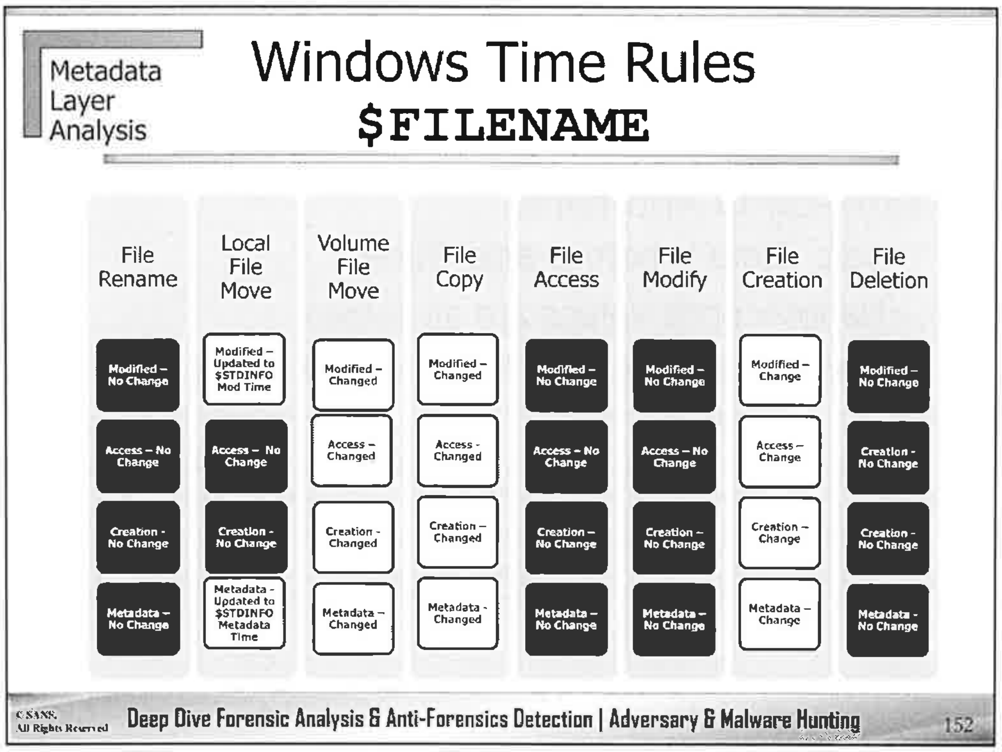


These are the general rules when it comes to files being moved, copied, accessed, modified, or created. The MFT change time is still being researched in regards to all of these times.

One note: When referring to the move command. We are generally referring to moving a file via "Cut-And-Paste" and not a move via a command line.

One note is that there is a significant difference between moving a file and copying a file for the creation time of a file. In addition, one of the many mistakes investigators make is assuming that disabling the last access time will result in zero updates to a file's last access time. This is incorrect. The file last access time will update during a file copy or move. It would not update by merely opening the file though.

Another note, while moving a file generally retains the Creation Date of the file. This is true if the action occurred via a "Cut-And-Paste" operation via Windows Explorer. If the move happened via the command prompt, the Creation Date changes.



This is the same tests performed while examining the $FILENAME Timestamps. The data here could help you determine when potential anti-forensics back dating of file timestamps might have occurred. The might be more than a single $Filename attribute per file. This would translate into an additional 4 timestamps for each additional $Filename attribute associated with a single MFT entry. Routinely, many files would have both a $Filename attribute for the long filename and short filename.

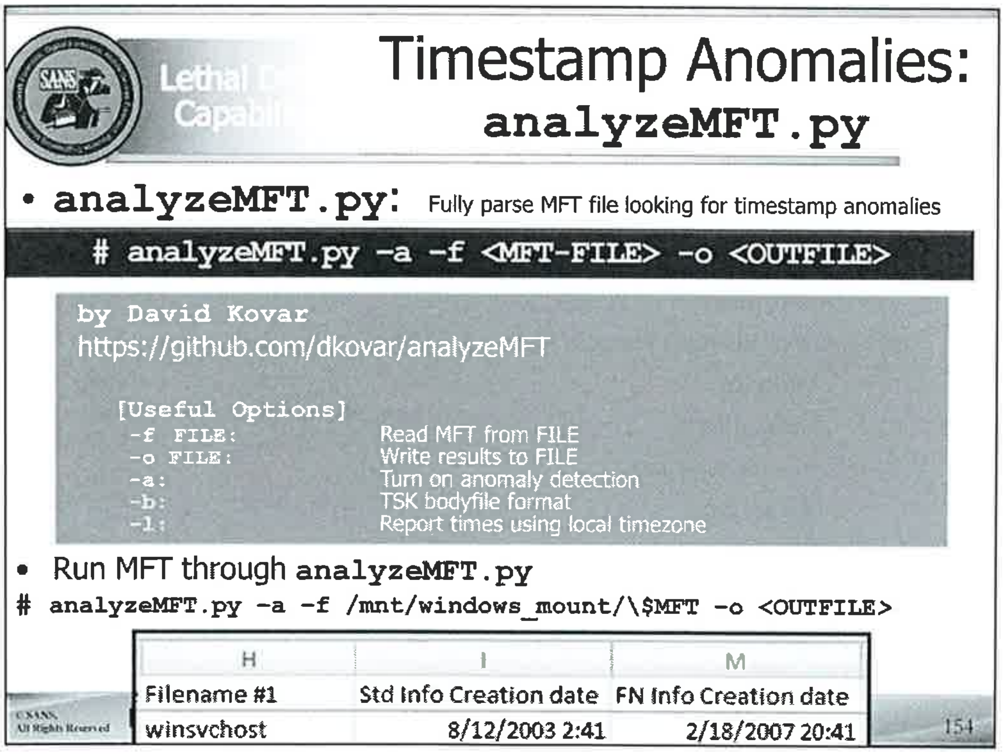
Detecting Anti-forensics Timestamp Anomalies:

- Timestamp Anomalies
  - **$SI** Time is before **$FN** Time
  - Nanoseconds values are all zeroes
  - Use istat to compare **$SI** vs. **$FN** times

H	I	M
Filename #1	Std Info Creation date	FN Info Creation date
winsvchost	8/12/2003 2:41	2/18/2007 20:41

One of the ways to tell if file time backdating occurred on a windows machine is to examine the filename times compared to the times stored in standard information. Tools such as timestomp allow a hacker to backdate a file to an arbitrary time of their choosing. Generally, hackers do this only to programs they are trying to hide in the system32 or similar system directories. Those directories and files would be a great place to start. You would look to see if the Filename (FN) time occurs after the Standard Info Creation Time as this would indicate an anomaly.

You can also compare this to the program compile time to see if the program was compiled after the creation date locally on the machine using the EXIFTool talked about in the "File Carving" Exercise.



Using a tool such as `analyzeMFT.py` will allow a investigator to quickly examine and compare filename times compared to standard information times.

`One other hint to look at. Every time an MFT record is re-used`
`analyzeMFT.py` is designed to fully parse the MFT file from an NTFS filesystem and present the results as accurately as possible in a format that allows further analysis with other tools. At present, it parses the attributes from a $MFT file to produce the following output:

- Record Number
- Good - if the entry is valid
- Active - if the entry is active
- Record type - the type of record

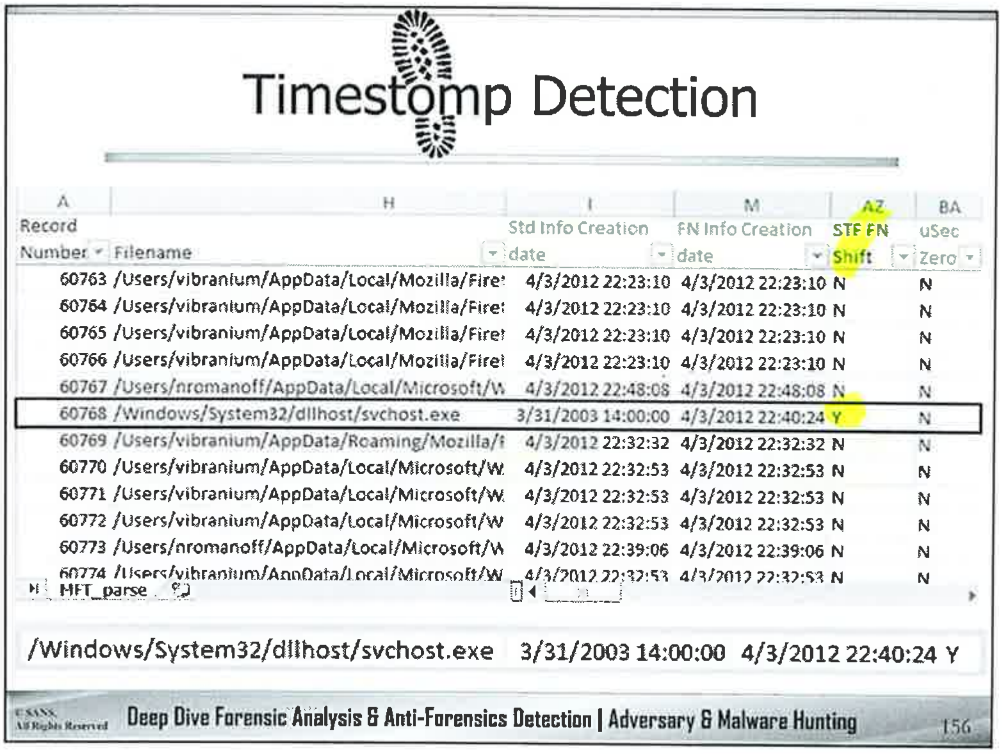
- Record Sequence - the sequence number for the record
- Parent Folder Record Number
- Parent Folder Sequence Number
- For the standard information attribute:
    - Creation date
    - Modification date
    - Access date
    - Entry date
- For up to four file name records:
    - File name
    - Creation date
    - Modification date
    - Access date
    - Entry date
- Object ID
- Birth Volume ID
- Birth Object ID
- Birth Domain ID

And flags to show if each of the following attributes is present:

Standard Information, Attribute List, Filename, Object ID, Volume Name, Volume Info, Data, Index Root, Index Allocation, Bitmap, Reparse Point, EA Information, EA, Property Set, Logged Utility Stream

Notes/Log - Field used to log any significant events or observations relating to this record.

For each entry in the MFT a record is written to an output file in CSV format.



Timestomp Detection

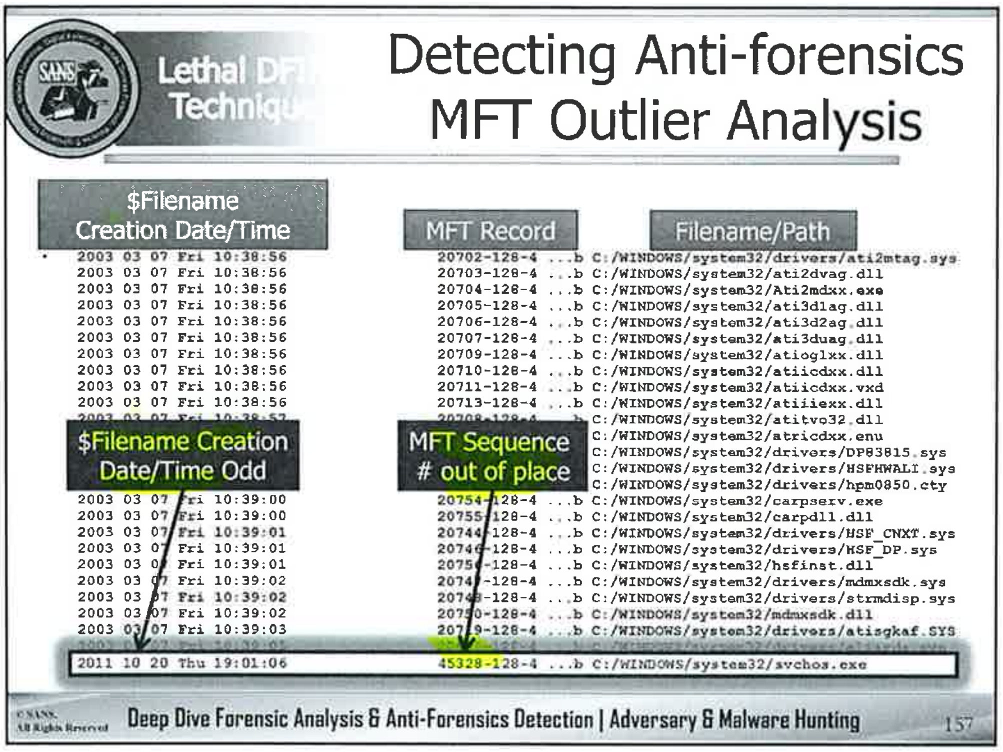
A Record Number	H Filename	I Std Info Creation date	M FN Info Creation date	AZ STF FN Shift	BA uSec Zero
60763	/Users/vibranium/AppData/Local/Mozilla/Fire:	4/3/2012 22:23:10	4/3/2012 22:23:10	N	N
60764	/Users/vibranium/AppData/Local/Mozilla/Fire:	4/3/2012 22:23:10	4/3/2012 22:23:10	N	N
60765	/Users/vibranium/AppData/Local/Mozilla/Fire!	4/3/2012 22:23:10	4/3/2012 22:23:10	N	N
60766	/Users/vibranium/AppData/Local/Mozilla/Fire!	4/3/2012 22:23:10	4/3/2012 22:23:10	N	N
60767	/Users/nromanoff/AppData/Local/Microsoft/W	4/3/2012 22:48:08	4/3/2012 22:48:08	N	N
60768	/Windows/System32/dllhost/svchost.exe	3/31/2003 14:00:00	4/3/2012 22:40:24	Y	N
60769	/Users/vibranium/AppData/Roaming/Mozilla/!	4/3/2012 22:32:32	4/3/2012 22:32:32	N	N
60770	/Users/vibranium/AppData/Local/Microsoft/W.	4/3/2012 22:32:53	4/3/2012 22:32:53	N	N
60771	/Users/vibranium/AppData/Local/Microsoft/W.	4/3/2012 22:32:53	4/3/2012 22:32:53	N	N
60772	/Users/vibranium/AppData/Local/Microsoft/W	4/3/2012 22:32:53	4/3/2012 22:32:53	N	N
60773	/Users/nromanoff/AppData/Local/Microsoft/W	4/3/2012 22:39:06	4/3/2012 22:39:06	N	N
60774	/Users/vibranium/AppData/Local/Microsoft/W	4/3/2012 22:32:53	4/3/2012 22:32:53	N	N

MFT_parse

/Windows/System32/dllhost/svchost.exe 3/31/2003 14:00:00 4/3/2012 22:40:24 Y

Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 156

This page intentionally left blank.



On NTFS the first file on the file system is assigned MFT Record Number zero and is always the $MFT. As more files are created on the system, the next file on the system would get MFT Record Number one, and so on. On EXT2/3/4 file systems, the root directory (/), is assigned MFT Record Number two and things increase from there as additional files are stored on the system.

A typical file system has hundreds of thousands of files. Each file has its own MFT Record Number. Because of the way operating systems are installed, it's normal to see entire directory structures written to disk with files having largely sequential MFT Record Number values.

So MFT Record Numbers are a metadata structure akin to a card (dating myself here) from a library's card catalog. They contain information about the files in the same way that those cards used to contain author, title, number of pages, location in the library, etc., but MFT Record Numbers contain owner, group, location on disk, size of file, etc. In a library these cards are arranged alphabetically either by title, author or subject. In a file system, they are simply first come, first serve and they are numbered sequentially. In NTFS MFT Record Number 0 always points to the $MFT.

Given that these MFT Record Numbers are assigned sequentially, if new files are written to disk, the MFT Record Numbers that are assigned to them are likely to be sequential or close to sequential, assuming a sequential run of MFT Record Numbers is available. I need to hire a good illustrator to animate this concept.

Think of it this way, as files are deleted from the system, their MFT Record Numbers are marked as unallocated and are available for reuse. If there are no unallocated MFT Record Numbers, new ones will be assigned beginning with the current maximum MFT Record Number value plus one and so on.

This will generally result in sequential MFT Record Number values, for example, here's a partial directory listing from a Windows NTFS partition's system32 directory (seen above).

This partial listing is sorted by date, note the MFT Record Number values that follow the date stamps are largely sequential, there are some exceptions, but for the most part the order of MFT Record Number numbers aligns with the file's creation times. As file systems are used over the years and new patches are applied causing files to be backed up and replaced, the ordering of these files by MFT Record Number numbers breaks down, but surprisingly, this ordering remains intact enough on many systems, even after years of use, that we can use them to spot files that may be of interest.

You essentially look for outliers that are outside the creation date/time and have a MFT Record Number that is not in alignment with the others in that directory. The best way to accomplish this in many cases is to create a scatter plot of the directory itself. But this technique is not well known or widely used, but not many hackers have implemented in anti-forensics defense against this as they are not thinking about how the way the file system will perform MFT Record Number allocation.

$Filename Creation Date/Time	MFT Record		Filename/Path
2003 03 07 Fri 10:38:56	20702-128-4	...b	C:/WINDOWS/system32/drivers/ati2mtag.sys
2003 03 07 Fri 10:38:56	20703-128-4	...b	C:/WINDOWS/system32/ati2dvag.dll
2003 03 07 Fri 10:38:56	20704-128-4	...b	C:/WINDOWS/system32/Ati2mdxx.exe
2003 03 07 Fri 10:38:56	20705-128-4	...b	C:/WINDOWS/system32/ati3d1ag.dll
2003 03 07 Fri 10:38:56	20706-128-4	...b	C:/WINDOWS/system32/ati3d2ag.dll
2003 03 07 Fri 10:38:56	20707-128-4	...b	C:/WINDOWS/system32/ati3duag.dll
2003 03 07 Fri 10:38:56	20709-128-4	...b	C:/WINDOWS/system32/atioglxx.dll
2003 03 07 Fri 10:38:56	20710-128-4	...b	C:/WINDOWS/system32/atiicdxx.dll
2003 03 07 Fri 10:38:56	20711-128-4	...b	C:/WINDOWS/system32/atiicdxx.vxd
2003 03 07 Fri 10:38:56	20713-128-4	...b	C:/WINDOWS/system32/atiiiexx.dll
2003 03 07 Fri 10:38:57	20708-128-4	...b	C:/WINDOWS/system32/atitvo32.dll
			C:/WINDOWS/system32/atricdxx.enu
			C:/WINDOWS/system32/drivers/DP83815.sys
			C:/WINDOWS/system32/drivers/HSFHWALI.sys
			C:/WINDOWS/system32/drivers/hpm0850.cty
2003 03 07 Fri 10:39:00	20754-128-4	...b	C:/WINDOWS/system32/carpserv.exe
2003 03 07 Fri 10:39:00	20755-128-4	...b	C:/WINDOWS/system32/carpdll.dll
2003 03 07 Fri 10:39:01	20744-128-4	...b	C:/WINDOWS/system32/drivers/HSF_CNXT.sys
2003 03 07 Fri 10:39:01	20746-128-4	...b	C:/WINDOWS/system32/drivers/HSF_DP.sys
2003 03 07 Fri 10:39:01	2075[illegible]-128-4	...b	C:/WINDOWS/system32/hsfinst.dll
2003 03 07 Fri 10:39:02	2074[illegible]-128-4	...b	C:/WINDOWS/system32/drivers/mdmxsdk.sys
2003 03 07 Fri 10:39:02	20748-128-4	...b	C:/WINDOWS/system32/drivers/strmdisp.sys
2003 03 07 Fri 10:39:02	207[illegible]0-128-4	...b	C:/WINDOWS/system32/mdmxsdk.dll
2003 03 07 Fri 10:39:03	207[illegible]9-128-4	...b	C:/WINDOWS/system32/drivers/atisgkaf.SYS
2003 03 07 Fri 10:39:05	[illegible]	...b	C:/WINDOWS/system32/drivers/[illegible]
2011 10 20 Thu 19:01:06	45328-128-4	...b	C:/WINDOWS/system32/svchos.exe

$Filename Creation Date/Time Odd

MFT Sequence # out of place

Metadata Layer Analysis

$Data

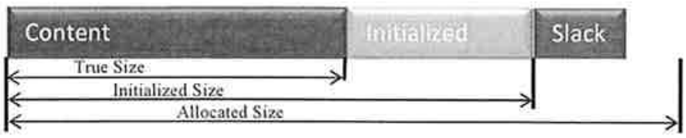
x00	80	00	00	00	48	00	00	00	01	00	00	00	00	00	03	00
	Attribute Signature				Attribute Length				Resident?							
x10	00	00	00	00	00	00	00	00	DC	00	00	00	00	00	00	00
	Starting VCN								Ending VCN							
x20	40	00	00	00	00	00	00	00	00	D0	0D	00	00	00	00	00
	Offset to Data Run								Allocated Size							
x30	00	CE	0D	00	00	00	00	00	00	CE	0D	00	00	00	00	00
	True Size								Initialized Size							
x40	31	DD	00	93	6B	00	FF	FF								
	Data Runs															

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 160

The $Data attribute is identified by the Signature value of x80 or 128.

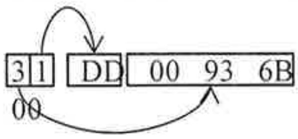
After the 24-byte attribute header, the content of the $Data attribute will contain either actual content or the information needed to figure out which clusters to read to find the data. A byte at offset x08 will mark if the content of this attribute is resident (x00) or nonresident (x01).

Virtual Cluster Numbers (VCN) are used to track the contiguous, in-order clusters that make up the file. Except in rare cases, these will always start at zero and end with a count of the number of clusters consumed by the content of the file. The Allocated Size is the size of the clusters consumed by the file (file content + file slack). The True Size is the actual size of the content of the file. The Initialized Size is the size of the clusters reserved for this file to grow into. Initialized Size is used when Windows allocates space for the file to grow into, but that space is not considered part of the actual content of the file, yet.



Data Runs consist of a series of entries that tell us the starting point and length of each segment of the file. Each entry has one-byte header that consists of two fields, one in each nibble. The 31 from the example above tells us that we need to read in 1 byte to get the length of the run and 3 bytes to get the starting cluster of that run. The next byte after that is x00, which tell us that is the end of the data runs.

Length of run (xDD clusters)



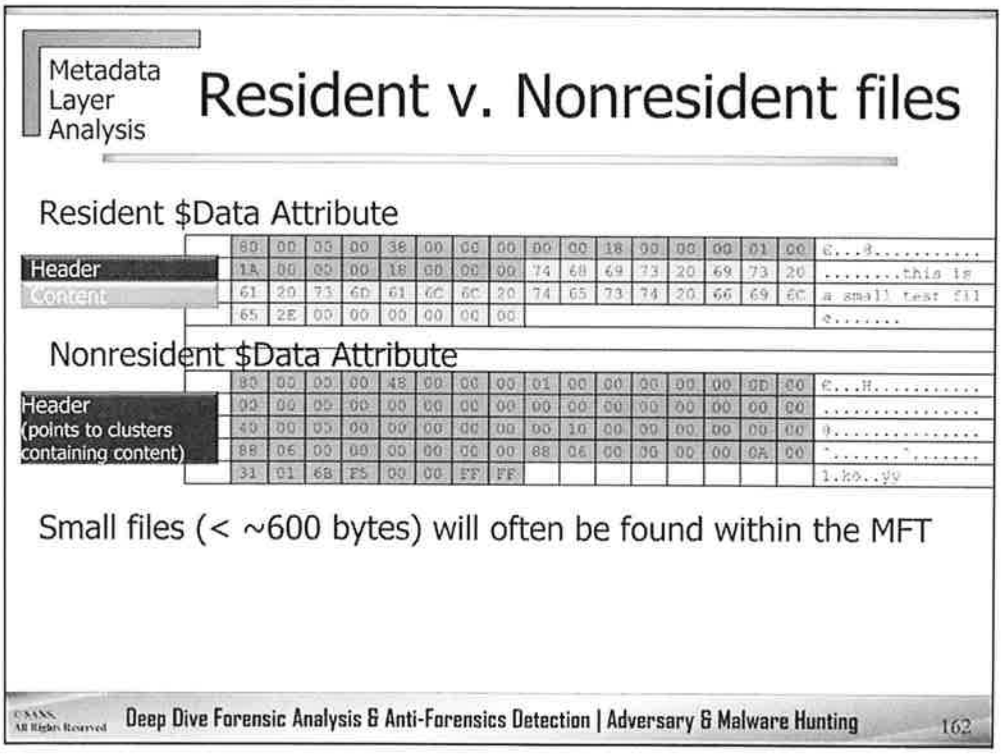
Offset to start of run (Cluster x6B9300)

Reference pages 355-359 of your **File System Forensic Analysis** book.

http://msdn.microsoft.com/en-us/library/windows/desktop/aa364404%28v=vs.85%29.aspx

$Data

x00	80	00	00	00	48	00	00	00	01	00	00	00	00	00	03	00
	Attribute Signature				Attribute Length				Resident?							
x10	00	00	00	00	00	00	00	00	DC	00	00	00	00	00	00	00
	Starting VCN								Ending VCN							
x20	40	00	00	00	00	00	00	00	00	D0	0D	00	00	00	00	00
	Offset to Data Run								Allocated Size							
x30	00	CE	0D	00	00	00	00	00	00	CE	0D	00	00	00	00	00
	True Size								Initialized Size							
x40	31	DD	00	93	6B	00	FF	FF								
	Data Runs															



The MFT Record for every file will contain at a minimum a record header, $Standard_Information, $File_Name, and $Data attributes. For a boring file that contains no other attributes and has a small name, this only consumes about 300 bytes, depending on length of the name. This leaves us as much as 720 bytes remaining in the MFT record that can be used to store the actual contents of the file. If the size is small enough, the $Data attribute will be resident and the content of the file will be right there. Once the file is too long to fit within the MFT record, the $Data attribute gets rewritten as a nonresident attribute and then contains a data run that tells the system which clusters to read in to find the content. If the MFT record needs to save a second $File_Name for a long file or if it contains any other attributes, such as $OBJECTID, this will shorten the amount of remaining space for the content.

When this happens, we end up with a $Data attribute like the one on top above. Starting at offset x18 will be the actual content of the file.

Another thing to take note of is that small files reside inside $MFT, which is an allocated file. If you are carving for deleted files in unallocated space, you will miss the files that are deleted but reside within the $MFT file.

Resident v. Nonresident files

Resident $Data Attribute

Header
Content

```
80 00 00 00 38 00 00 00 00 00 18 00 00 00 01 00  €...8...........
1A 00 00 00 18 00 00 00 74 68 69 73 20 69 73 20  ........this is 
61 20 73 6D 61 6C 6C 20 74 65 73 74 20 66 69 6C  a small test fil
65 2E 00 00 00 00 00 00                          e.......
```

Nonresident $Data Attribute

Header
(points to clusters containing content)

```
80 00 00 00 48 00 00 00 01 00 00 00 00 00 0D 00  €...H...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
40 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00  @...............
88 06 00 00 00 00 00 00 88 06 00 00 00 00 0A 00  ˆ.......ˆ.......
31 01 6B F5 00 00 FF FF                          1.kõ..ÿÿ
```

Small files (< ~600 bytes) will often be found within the MFT

Metadata Layer Analysis

Alternate Data Streams (ADS)

x00									80	00	00	00	30	00	00	00	€...0...
x10	00	00	18	00	00	00	01	00	18	00	00	00	18	00	00	00	....
x20	54	68	69	73	20	69	73	20	61	20	6E	6F	72	6D	61	6C	This is a normal
x30	20	66	69	6C	65	20	0D	0A									file ..
									80	00	00	00	50	00	00	00	€...P...
x40	00	08	18	00	00	00	03	00	23	00	00	00	28	00	00	00	........#...(...
x50	41	00	6C	00	74	00	20	00	44	00	61	00	74	00	61	00	A.l.t. .D.a.t.a.
x60	54	68	69	73	20	69	73	20	61	6E	20	61	6C	74	65	72	This is an alter
x70	6E	61	74	65	20	64	61	74	61	20	73	74	72	65	61	6D	nate data stream

Name

ADS is alternate content for a file, that exists by creating additional **`$Data`** attributes within the same MFT record

 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 164

'Cause sometimes one just isn't enough.

Originally included in Windows NT as part of the Services for Macintosh functionality as a way for a Windows file server to support the concept of a Resource Fork that exists in Mac file systems. Ironically, Macintosh clients don't make use of it and opt instead to create a second hidden file that contains the second set of data.

Alternate Data Streams are merely the presence of a second or subsequent $Data attribute, much in the same vein that a file can have multiple $File_Name attributes. The primary attribute is never named, deriving its name from the $File_Name attribute. Any subsequent $Data attributes must be named in order to able to address them. This is accomplished by including a Unicode string just before the content of the ADS for resident attributes, and immediately preceding the data runs for nonresident attributes.

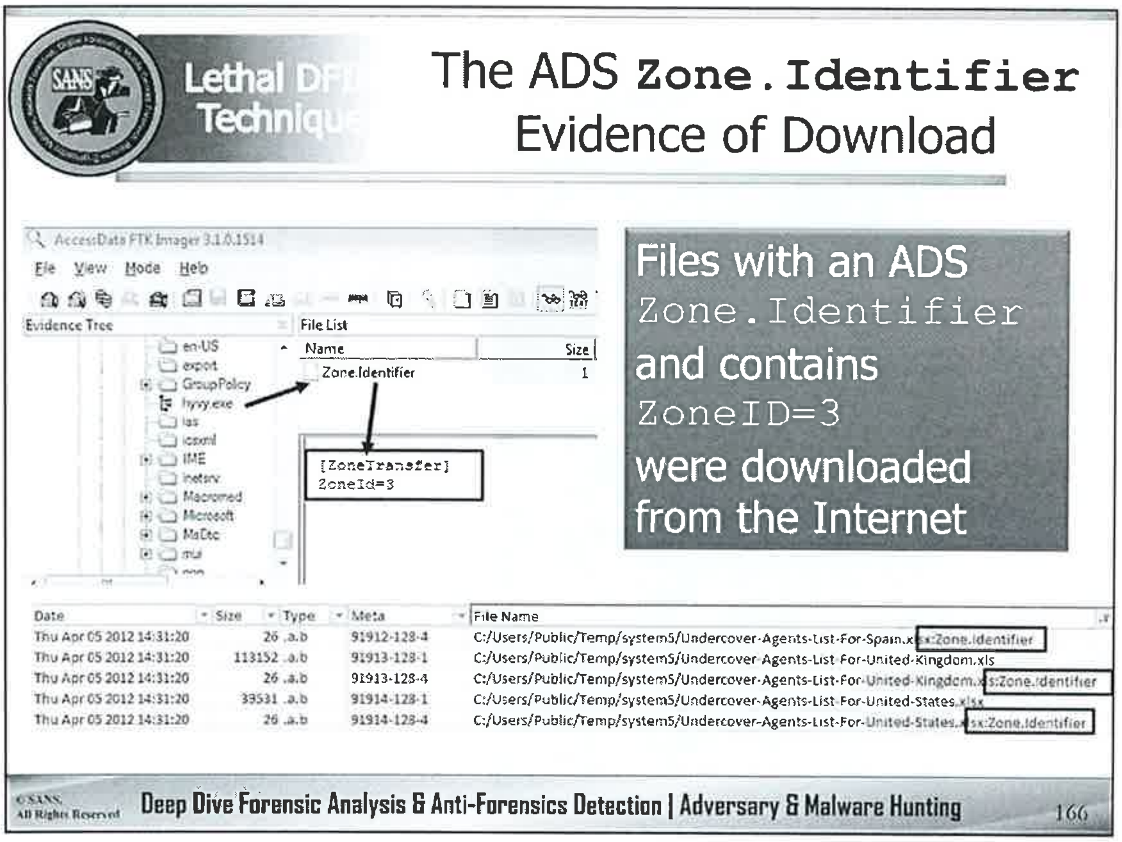
Since the rest of Windows is only designed to accommodate the primary data stream, most tools cannot report on the existence of an alternate data stream, let alone its size and contents. No other metadata about an alternate stream is maintained. This makes ADS an attractive place to hide illicit tools or data. Each version of Windows from XP SP3 to 7 has placed more and more restrictions on the things you can do with an ADS, making them less useful to hackers.

Alternate Data Streams

Name

```
x00                                  80 00 00 00 30 00 00 00         €...0...
x10  00 00 18 00 00 00 01 00 18 00 00 00 18 00 00 00 ................
x20  54 68 69 73 20 69 73 20 61 20 6E 6F 72 6D 61 6C This is a normal
x30  20 66 69 6C 65 20 0D 0A                           file ..
                                     80 00 00 00 50 00 00 00         €...P...
x40  00 08 18 00 00 00 03 00 23 00 00 00 28 00 00 00 ........#...(...
x50  41 00 6C 00 74 00 20 00 44 00 61 00 74 00 61 00 A.l.t. .D.a.t.a.
x60  54 68 69 73 20 69 73 20 61 6E 20 61 6C 74 65 72 This is an alter
x70  6E 61 74 65 20 64 61 74 61 20 73 74 72 65 61 6D nate data stream
```

ADS is alternate content for a file, that exists by creating additional $Data attributes within the same MFT record

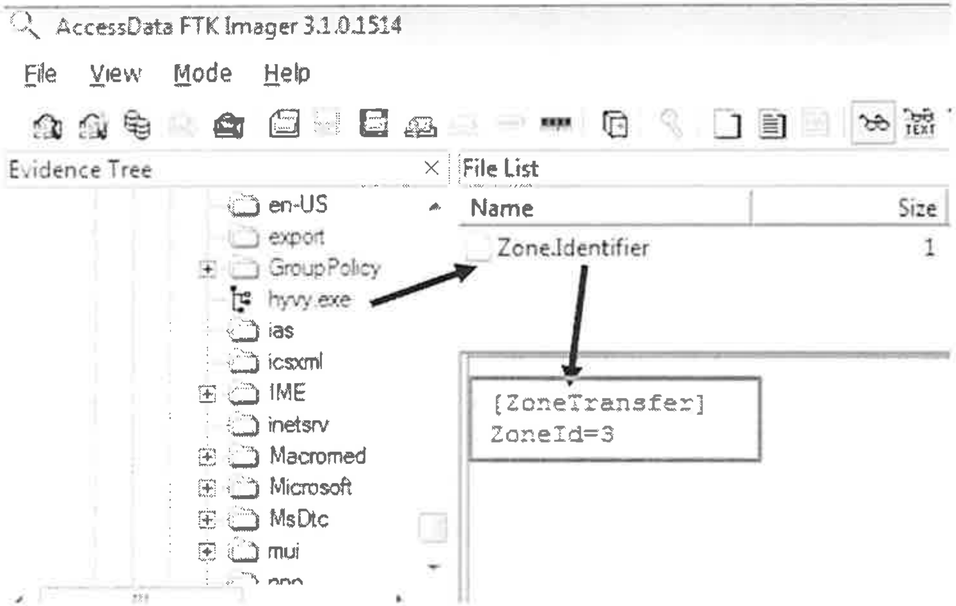


Starting with XP SP2 when files are downloaded from the Internet via a browser to a NTFS volume an alternate data stream is added to the file. The alternate data stream is named "Zone.Identifier." The Zone.Identifier usually has simple text with the stream "ZoneID=3." Value 3 means that the file was downloaded from the Internet and it's potentially unsafe. The values could be from other zones. They are listed below:

NoZone = -1 MyComputer = 0 Intranet = 1 Trusted = 2 Internet = 3 Untrusted = 4

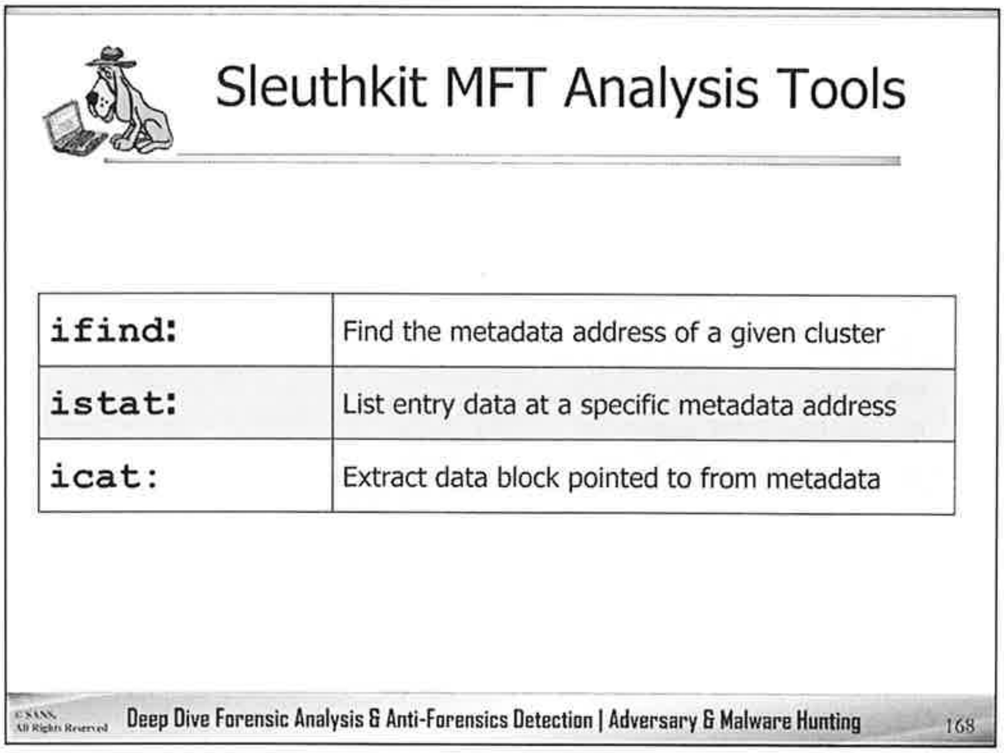
The key analysis point here is that if you have ever wondered how a word document or powerpoint that you open has been tagged as "downloaded" from the Internet and where you have to "Enable" it to edit it, then you might see the effect of the ZoneID=3 value every day. From an analysis perspective, ANY file that has the Zone.Identifier with ZoneID=3 value originated from the Internet and is direct evidence of "File Download".

This can be useful as not only document and media files are affected by this. We can see above that some malware might have this trigger as well. It is useful to quickly scan C:\Windows\System32 to see if any files have the "Zone.Identifier" ADS. It could be evil.



Files with an ADS
`Zone.Identifier`
`ZoneID=3`
were downloaded from the internet

Date	Size	Type	Meta	File Name
Thu Apr 05 2012 14:31:20	26	.a.b	91912-128-4	C:/Users/Public/Temp/system5/Undercover-Agents-List-For-Spain.xlsx:Zone.Identifier
Thu Apr 05 2012 14:31:20	113152	.a.b	91913-128-1	C:/Users/Public/Temp/system5/Undercover-Agents-List-For-United-Kingdom.xls
Thu Apr 05 2012 14:31:20	26	.a.b	91913-128-4	C:/Users/Public/Temp/system5/Undercover-Agents-List-For-United-Kingdom.xls:Zone.Identifier
Thu Apr 05 2012 14:31:20	39531	.a.b	91914-128-1	C:/Users/Public/Temp/system5/Undercover-Agents-List-For-United-States.xlsx
Thu Apr 05 2012 14:31:20	26	.a.b	91914-128-4	C:/Users/Public/Temp/system5/Undercover-Agents-List-For-United-States.xlsx:Zone.Identifier



In the next few slides, we will cover some of the key meta data examination tools.

ifind overview

- Frequently, one will find interesting data in a block by keyword searching or other means
- You could probably learn more if you knew which metadata structure allocated it ...
- The `ifind` tool searches the metadata structures to find one that points to the data unit
- It is used most frequently when doing keyword searches

Usage:

```
ifind [options] [-d unit_addr] [-n file] [-p par_addr] image
```

[Options for istat]

`-a:`	Find all inodes
`-d unit_addr:`	Find the meta data given the data unit
`-n file:`	Find the meta data given the `file name`
`-p par_addr:`	Find UNALLOCATED MFT entries given the parent's meta address (NTFS only)

As we will later see, keyword searches are important for recovering deleted content. Using 'grep', we can get the byte offset of the string. Then we can divide the byte offset by the size of the data unit to find out which data unit it is part of.

The ifind tool then searches the metadata structures to find the one that points to the data unit of interest. That way, we can examine the entire file that has this certain string. Also, with some file systems, the data unit pointer is saved when the file is deleted. The ifind tool will even identify unallocated inodes for those systems.

icat overview:

- Copies files by inode number
- Capabilities
  - Writes files specified by inode number to stdout
  - Can skip over holes in files
- Operates on
  - disk device
  - file containing a disk image
- Useful for recovering deleted files

Usage:

`icat [options] image inode`

`[Options for icat]`

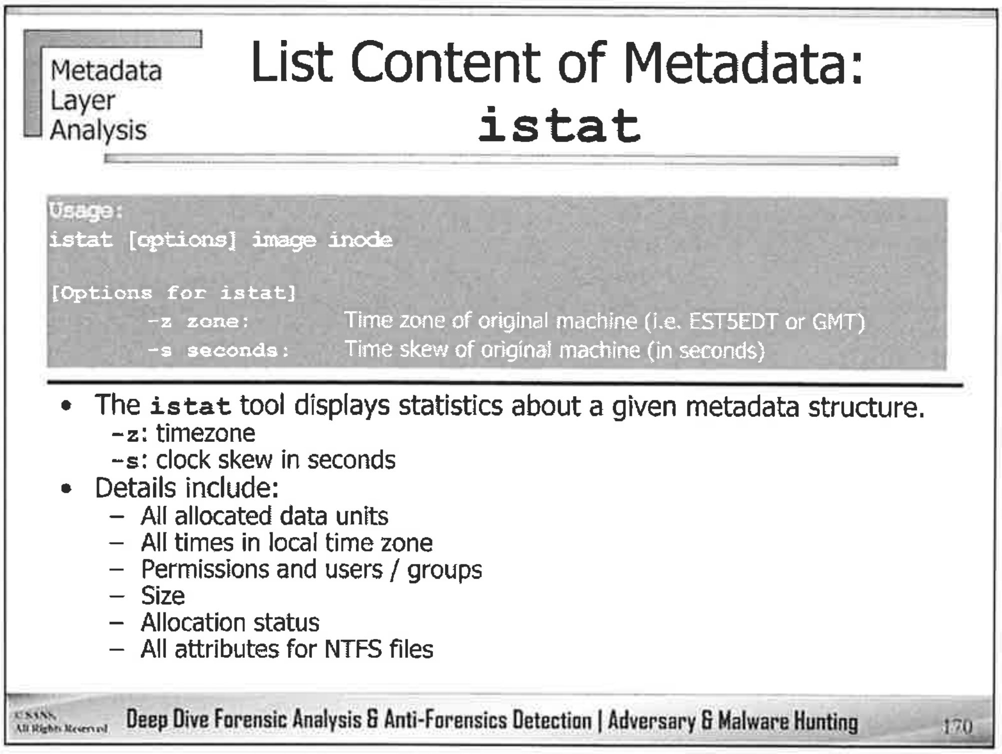
`-r:`	Recover deleted file
`-R:`	Recover deleted file and suppress recovery errors
`-s:`	Display slack space at end of file

Given an inode (particularly of a deleted file) it is often desirable to copy the file by its inode number. The icat utility does just this. It accesses the file specified by a given inode and writes it to the stdout file descriptor. Obviously, this can be redirected to a file. As with the other commands, this operation can be performed on an actual disk device, or on a file that contains a disk image. The most common use for icat is to recover deleted files, or at least portions of them.

With flags, you can force 'icat' to ignore holes (which are blocks of all zeros).

"The '-s' flag will force 'icat' to also output the slack space of a file. For example, consider a file that is 2000 bytes in size and has a 4096 byte cluster allocated to it. Normally, 'icat' will only output the 2000 bytes, but with '-s', then 'icat' will output the full 4096 bytes from the cluster.

The '-r' flag causes 'icat' to perform file recovery for some file systems, specifically with FAT. Some file systems, such as NTFS, do not need any special steps to recover deleted files. Others, such as FAT do. Using the '-r' flag causes 'icat' to make assumptions about where the original data is located and the data may not be accurate (because the original file was fragmented or because part of the file was overwritten).



Usage:

```
istat [options] image inode

[Options for istat]
        -z zone:      Time zone of original machine (i.e. EST5EDT or GMT)
        -s seconds:   Time skew of original machine (in seconds)
```

The 'istat' tool displays all details known about a given file. This dumps almost all of the data in the metadata structure, and then some. This is an example of how The Sleuth Kit has been designed to show you all of the information it can find.

This is useful to learn which blocks an inode has allocated and the times for the last modification, access, and change.

The istat tool for NTFS images will show details about every attribute in a file.

The major flags of istat deal with time issues. The -z flag allows you to specify the time zone of where the image was acquired from (otherwise the local time zone will be used). The '-s' flag allows you to specify the clock skew of the system. For example, if you knew that the system was slow by 32 seconds then the flag '-s -32' would be given. Then, both the adjusted and un-adjusted times would be displayed.

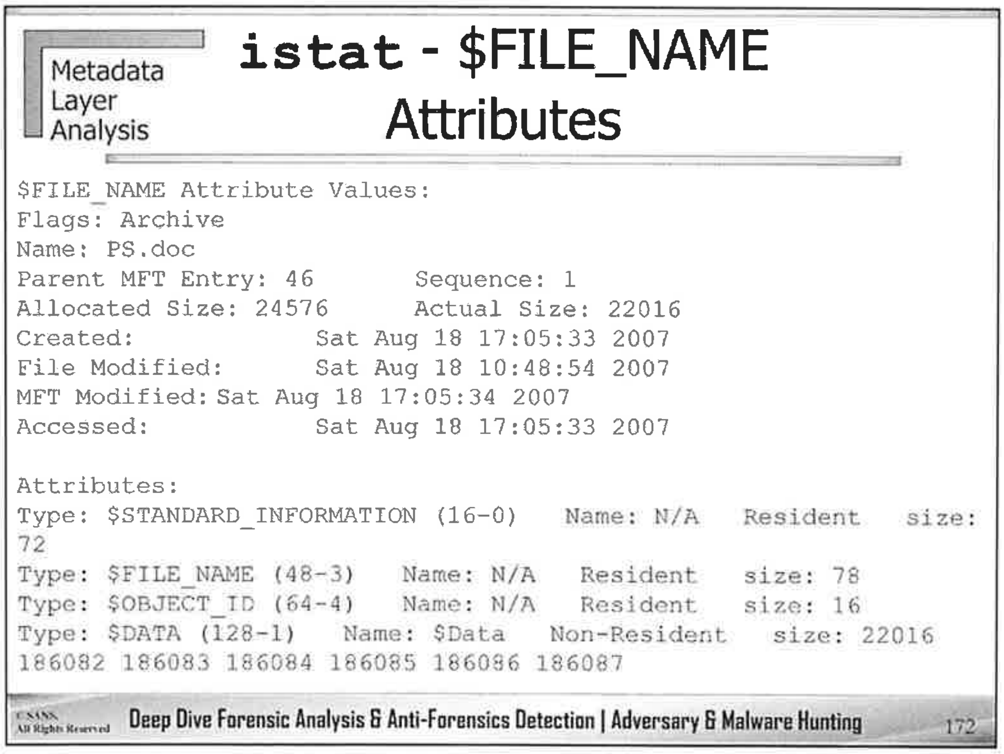
Metadata Layer Analysis

istat - $STD_INFO Attributes

```
# istat evidence.img 60
MFT Entry Header Values:
Entry: 60        Sequence: 3
$LogFile Sequence Number: 6781202
Not Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 272  ()
Last User Journal Update Sequence Number: 97216
Created:        Sat Aug 18 17:05:33 2007
File Modified:  Sat Aug 18 10:48:54 2007
MFT Modified:   Sat Aug 18 18:41:51 2007
Accessed:       Sat Aug 18 17:05:33 2007
```

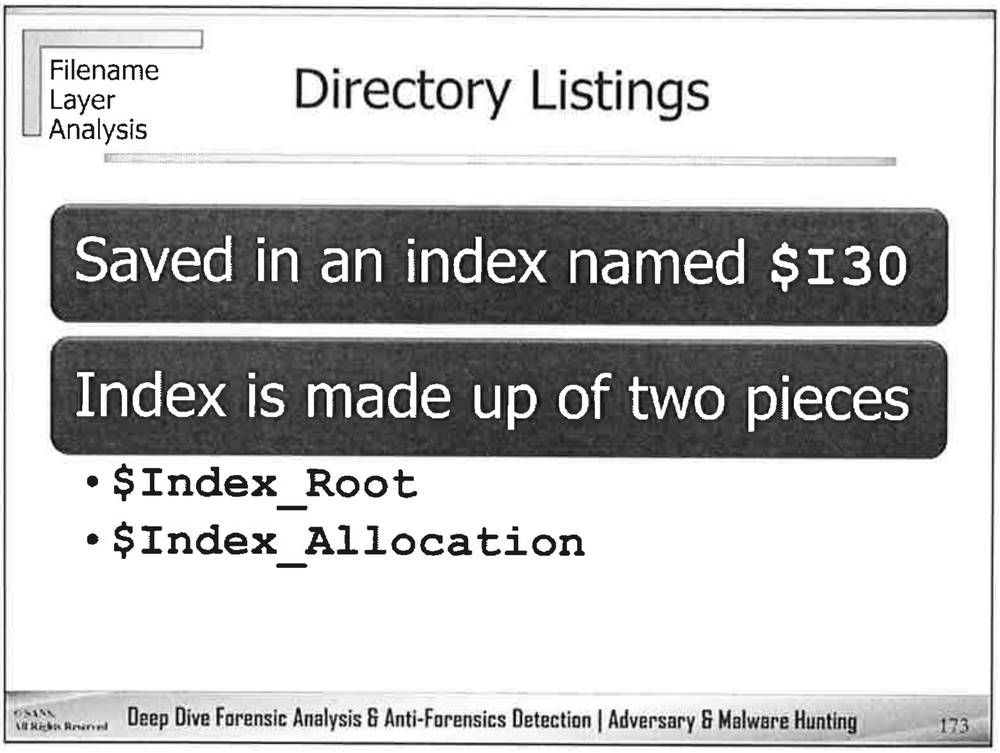

Here we see the output from running 'istat' on an NTFS file. NTFS is a more complex file system than FAT and 'istat' prints a lot more information for an NTFS image than a FAT image. The output is broken up into two slides.

The top part shows the general metadata for the file, including its address and allocation status. Files have a series of attributes and each attribute contains specific information about the file. The second part of the output contains information in the $STANDARD_INFORMATION attribute, which includes the temporal data for the file. There are two more parts to the output and they are on the next slide.



The third and fourth parts of the istat output on an NTFS image are shown in this slide. The $FILE_NAME attribute contains information about the file's name, parent directory, and more time values. The time values were removed from the slide so that it would all fit.

Lastly, all of the attributes are given. The $DATA attribute is where the file content is stored and we see that it is 72 bytes in this example.



These Indexes consume an MFT record and contain $Standard_Information and $File_Name attributes just like any file. The difference is that instead of a $Data attribute, it contains a structured index that lists the contents of the directory.

Just like files can be resident or nonresident depending on their size, directory listings can be either resident or nonresident depending on the number of files they contain.

The index is made up of two constructs working together. $Index_Root contains the information about the size of the entries, how to sort them, and such. If there are only a few entries, then they will be listed in this attribute and be resident inside the MFT. If there are more than a few, then the $Index_Root is just the index header and a second attribute, $Index_Allocation, is used to store the actual index entries. $Index_Allocation attributes will thus always be nonresident and are stored as almost exactly the same as $Data attributes from a few pages ago with data runs to describe which clusters contain the index.

The index is named $I30. Remember what the attribute signature of a $File_Name attribute is? X30 – that should give you a clue how to read the entries in this index. The $File_Name attribute listed a few pages ago included an attribute header that was 24 bytes long – remove that header and replace it with a 16 byte Index Entry header, and the rest of the structure is identical.

The index gets rewritten fairly regularly as an update to any file within the directory causes the index to be reshuffled to ensure proper sorting and updated metadata. Thus, the time stamps of the entries here usually match the $Standard_Information attribute of the referenced file.

Usage:
`ffind [options] image inode`

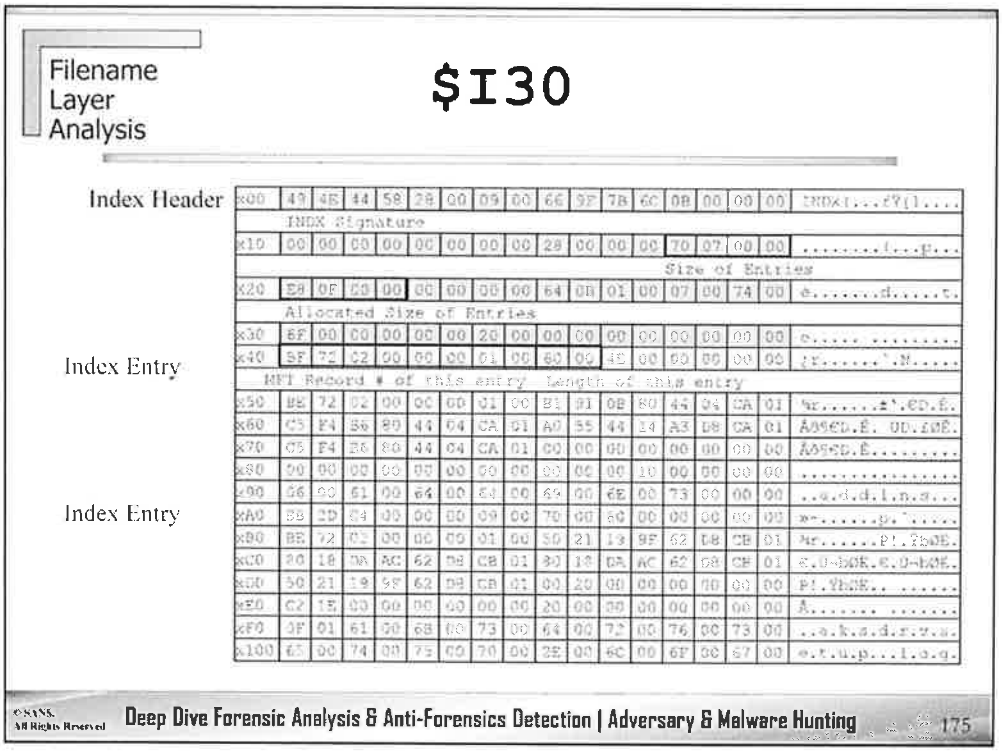
[Useful Options for ffind]

- **`-a:`** Find all occurrences
- **`-r:`** Recurse on directories

The **`ffind`** tool is a mapping tool that finds the file name for a metadata address. This is useful when we find a metadata structure using ifind and still want to get more context. The tool processes the full directory tree and looks for an entry that points to the metadata address

The ffind tool is the complement to the ifind tool. The ifind tool found a metadata structure that was pointing to a data unit. The ffind tool finds a file name that is pointing to a metadata structure. Therefore, if we do a keyword search, we can tie the keyword to a file using these tools and the byte offset of the string.

For file systems that do not delete the pointer from the file name structure to the metadata structure, the deleted file name can be found.



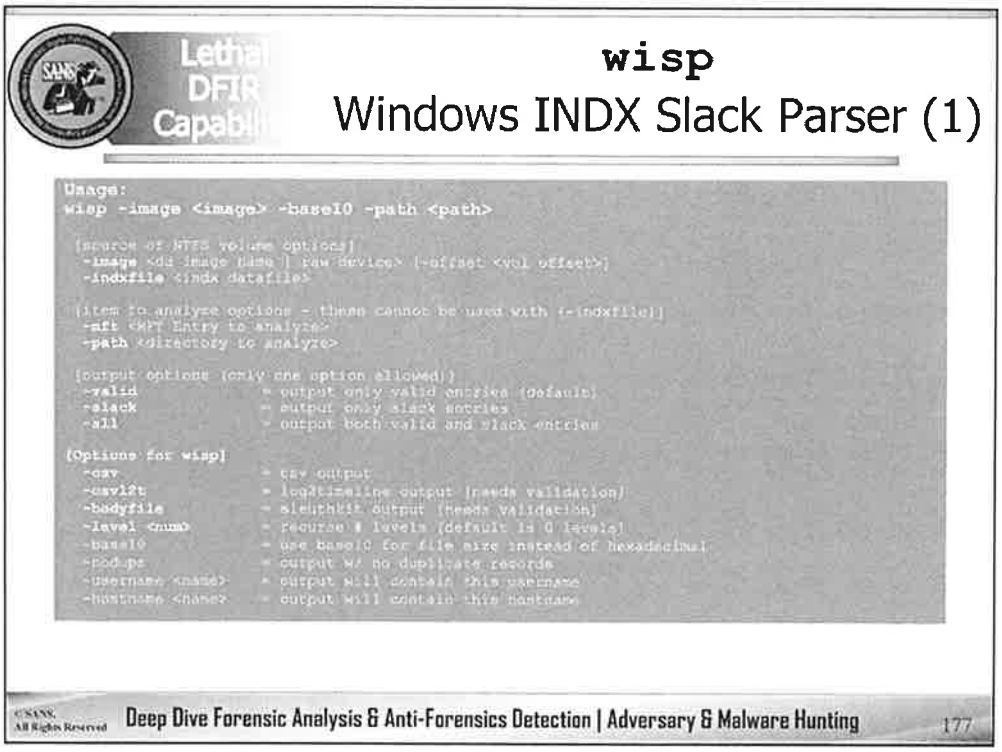
Each Index Block will start with the signature "INDX" followed by a short header. The most important fields of this header to us are the two size fields at offset x1C and x20. The Allocated Size is the size of the Index Block minus part of the header. Index Blocks are always the same as the cluster size, so they should usually be 4096 bytes. The other size field, the one at x1C, is the more important one. It tells us how much of the Index Block is actually in use. If you only have a couple of files in a directory, their entries will only take up a few hundred bytes, but Index Blocks are allocated in chucks of the cluster size (4096 bytes). That means slack space. If you have a large directory and you delete a bunch of files, the index will get rewritten with a shorter list, and there will be orphaned Index Entries in that slack space. That doesn't just happen in the last Index Block either. Every block will have some amount of slack space that may or may not have some remnant information in it.

Each Index Entry will have a short 16-byte header. The important parts of this header are the MFT Record Number of the file referenced by this entry and a length field that tells us how big this entry is, and thus where the start of the next entry will be. The rest of the Index Entry, starting at offset x50 in the example above and highlighted, will parse out exactly the same as a $File_Name attribute, from Parent Directory Reference, four time stamps, size, flags, and the file name. If the file has two $File_Name attributes to accommodate long and short names, there will be two Index Entries, one for each name.

In summary: Directory listings are stored in an Index. If the index is short, you will find the Index Entries described above trailing at the end of the $Index_Root attribute. Once there are enough entries that they won't all fit inside the MFT record, and $Index_Allocation attribute is created and contains the Index Blocks described above. Indexes are made up of multiple Index Blocks that are allocated as necessary and are the same size as the cluster size. Each Index Block will have the Index Header like the one in the darker highlight in the example above, followed by multiple Index Entries. Each Index Entry will be read like reading a $File_Name attribute with the addition of referenced file's MFT Record Number at the start of the entry.

$I30

```
Index Header x00  49 4E 44 58 28 00 09 00 66 9F 7B 6C 0B 00 00 00  INDX(...fY{l....
                  INDX Signature
             x10  00 00 00 00 00 00 00 00 28 00 00 00 70 07 00 00  ........(...p...
                                                      Size of Entries
             X20  E8 0F 00 00 00 00 00 00 64 0B 01 00 07 00 74 00  è.......d.....t.
                  Allocated Size of Entries
             x30  6F 00 00 00 00 00 20 00 00 00 00 00 00 00 00 00  o..... .........
Index Entry  x40  BF 72 02 00 00 00 01 00 60 00 4E 00 00 00 00 00  ¿r......`.N.....
                MFT Record # of this entry  Length of this entry
             x50  BE 72 02 00 00 00 01 00 B1 91 0B 80 44 04 CA 01  ¾r......±'.€D.Ê.
             x60  C5 F4 B6 80 44 04 CA 01 A0 55 44 14 A3 D8 CA 01  Åô¶€D.Ê. UD.£ØÊ.
             x70  C5 F4 B6 80 44 04 CA 01 00 00 00 00 00 00 00 00  Åô¶€D.Ê.........
             x80  00 00 00 00 00 00 00 00 00 00 00 10 00 00 00 00  ................
             x90  06 00 61 00 64 00 64 00 69 00 6E 00 73 00 00 00  ..a.d.d.i.n.s...
Index Entry  xA0  BB 2D 04 00 00 00 09 00 70 00 60 00 00 00 00 00  »-......p.`.....
             xB0  BE 72 02 00 00 00 01 00 50 21 19 9F 62 D8 CB 01  ¾r......P!.YbØË.
             xC0  80 18 DA AC 62 D8 CB 01 80 18 DA AC 62 D8 CB 01  €.Ú¬bØË.€.Ú¬bØË.
             xD0  50 21 19 9F 62 D8 CB 01 00 20 00 00 00 00 00 00  P!.YbØË.. ......
             xE0  C2 1E 00 00 00 00 00 00 20 00 00 00 00 00 00 00  Â....... .......
             xF0  0F 01 61 00 6B 00 73 00 64 00 72 00 76 00 73 00  ..a.k.s.d.r.v.s.
             x100 65 00 74 00 75 00 70 00 2E 00 6C 00 6F 00 67 00  e.t.u.p...l.o.g.
```



Used with permission from Tzworks, LLC –

wisp is a prototype version of a Windows parser that targets NTFS index type attributes. The NTFS index attribute points to one or more INDX records. These records contain index entries that are used to account for each item in a directory. An index item represents either a file or a subdirectory and includes enough metadata to contain the name, modified/access/MFT changed/birth (MACB) timestamps, size (if it is a file vs. subdirectory), as well as MFT entry numbers of the item and its parent. The wisp tool, in its simplest form, is able to walk these structures, read the metadata, and report which index entries are present.

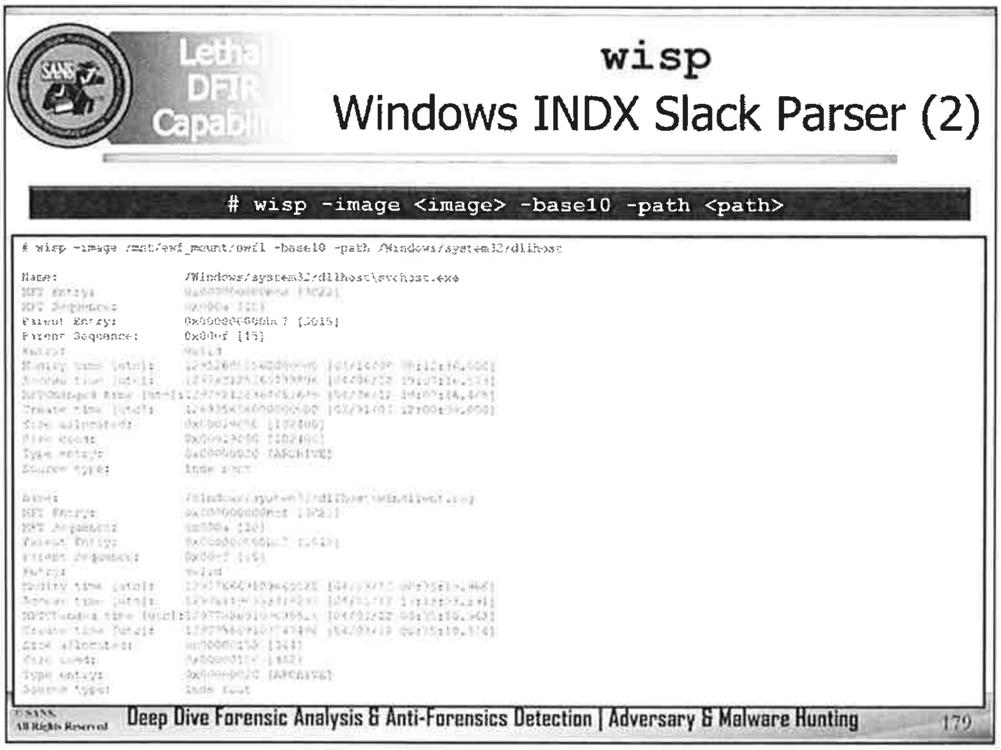
As a directory's contents are changed, the number of valid index entries grows or shrinks, as appropriate. As more directory entries are added, eventually it will exceed the existing INDX record allocation space. At this point the operating system will allocate an additional INDX record in the size of 0x1000 byte chunk. Conversely, when entries are removed from the directory, the INDX record space is not necessarily deallocated. Thus, anytime the number of index entries shrinks, the invalid ones potentially can be harvested from the slack space.

VERY IMPORTANT: The slack space is defined to be the allocated but unused space. By comparing both the valid entries and those still in the slack space, one can make some inferences about whether a file (or subdirectory) was present in the past.

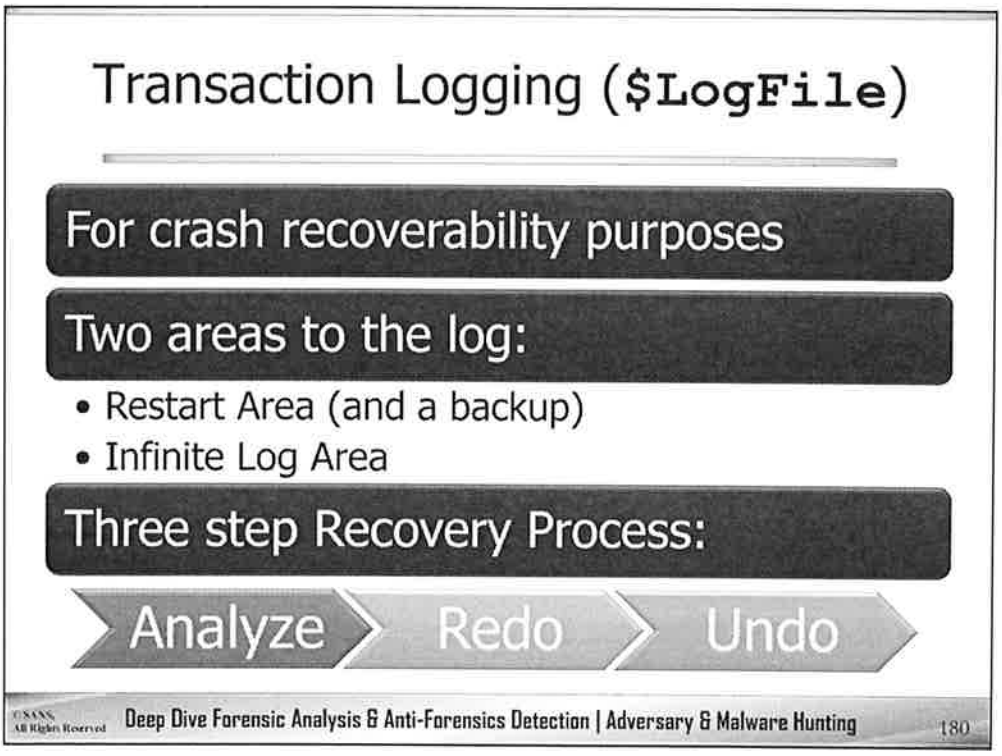
A good tutorial on harvesting index entries from INDX slack space can be found on Willi Ballenthin's webpage [1] and his DFIRonline presentation [2].

References
[1] - http://www.williballenthin.com/forensics/indx/index.html
[2] - http://writeblocked.org/dfironline.html

```
Usage:
wisp -image <image> -base10 -path <path>

 [source of NTFS volume options]
  -image <dd image name | raw device> [-offset <vol offset>]
  -indxfile <indx datafile>

 [item to analyze options - these cannot be used with (-indxfile)]
  -mft <MFT Entry to analyze>
  -path <directory to analyze>

 [output options (only one option allowed)]
  -valid                = output only valid entries (default)
  -slack                = output only slack entries
  -all                  = output both valid and slack entries

[Options for wisp]
  -csv                  = csv output
  -csvl2t               = log2timeline output [needs validation]
  -bodyfile             = sleuthkit output [needs validation]
  -level <num>          = recurse # levels [default is 0 levels]
  -base10               = use base10 for file size instead of hexadecimal
  -nodups               = output w/ no duplicate records
  -username <name>      = output will contain this username
  -hostname <name>      = output will contain this hostname
```




```
usage: INDXParse.py [-h] [-c | -b] [-d] filename

Parse NTFS INDX files.

positional arguments:
  filename     Input INDX file path

optional arguments:
  -h, --help   show this help message and exit
  -c           Output CSV
  -b           Output Bodyfile
  -d           Find entries in slack space
```

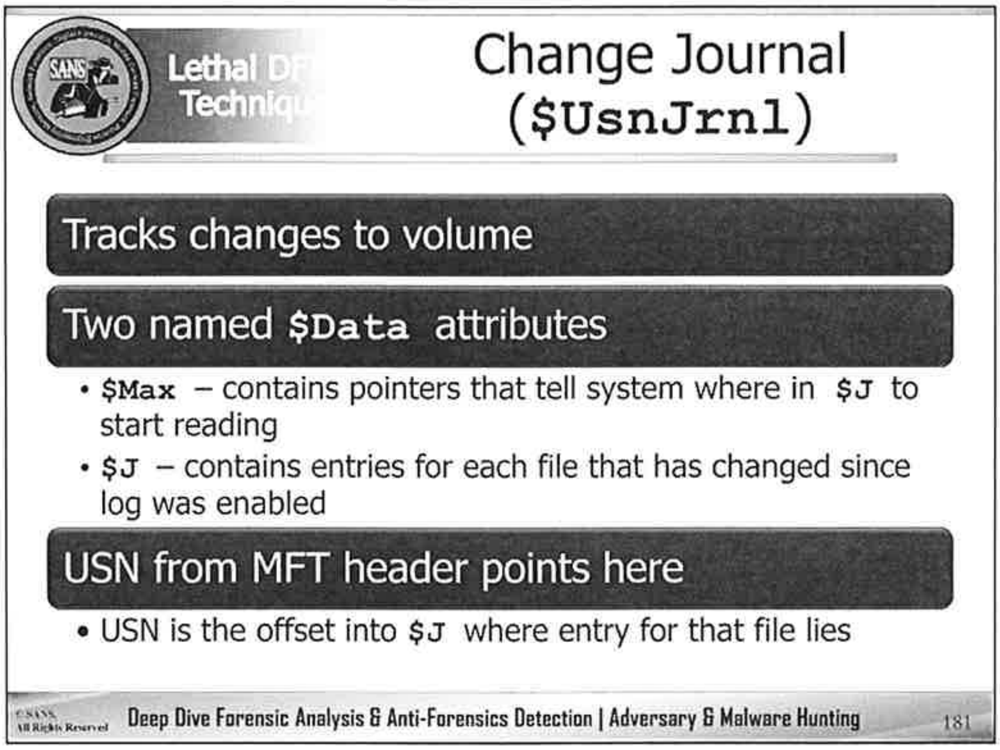


NTFS uses the $LogFile to maintain a log of all file transactions, such as a read, write, or modification of any file. It takes multiple steps to update multiple structures in order to write to a single file; the $logFile keeps track of each of those steps so the state of each transaction can be tracked. Once a transaction is complete, NTFS commits the file update to disk. If there is an error or if there is any reason for NTFS to believe the transaction did not complete, NTFS rolls back the steps that did occur according to undo information stored in the log. Incomplete modifications to the volume are not allowed. Tracks file by the $LogFile Sequence Number found in MFT Record header.

The actual internals of the $LogFile consist of two structures: A **Restart Area**, with a signature value of RSTR. There will be two of these in case the first one is unreadable for any reason. This area contains information about the location and contents of the actual log. Next is an **Infinite Log Area**, which is a circular log file that is constantly being reused, which consists of a series of records. The actual data is stored in records with a signature of RCRD. **Update Records** are the most common and contain both undo and redo information – essentially, it contains a before and after state for every change. There are also **Checkpoint Records** stored every 8 seconds. These records provide a starting point for the recovery process – the system should only have to recover from most recent checkpoint rather than having to analyze the entire log.

If the system crashes, NTFS performs three passes through the data on the disk: The **Analysis pass** examines the last checkpoint record in the log. At the end of this pass, the transaction table contains only transactions that were active at the time of the crash. The **Redo pass** performs any steps logged from the last checkpoint so that the cache reflects the state of the volume from when the crash occurred. Lastly, the **Undo pass** rolls back any incomplete transactions in order to return the volume to a stable state. This process does not guarantee that no user data will be lost. It does ensure that all the various file records, indexes, logs, and various other structures return to a consistent state after a crash so that a crash doesn't take down whole system.

http://technet.microsoft.com/en-us/library/cc976815.aspx
http://support.microsoft.com/kb/101670



Internally it is called the Update Sequence Number (USN) Journal (thus the file name of $USNJrnl), but all of the marketing and user manuals use the name Change Journal. Whenever there is a change to the volume an entry is created in this log. This is very useful for utilities that need to do incremental scans of a system, such as backup or anti-virus programs. The USN value is found in each file's $Standard_Information attribute. It is used as an offset into the $J file (the actual USN Journal) and identifies the last change made to that specific file.

Tracked in the record is file's MFT number, the file's parent directory's MFT number, a timestamp, a reason code that tells us what about the file changed, the file's size, file's attributes, and the file's name.

0x00 4 Size of entry

0x04 2 Major Version

0x06 2 Minor Version

0x08 8 MFT Reference

0x10 8 Parent MFT Reference

0x18 8 Offset of this entry in $J

0x20 8 Timestamp

0x28 4 Reason (see below)

0x2B 4 SourceInfo

0x30 4 SecurityID

0x34 4 FileAttributes

0x38 2 Size of filename (in bytes)

0x3A 2 Offset to filename

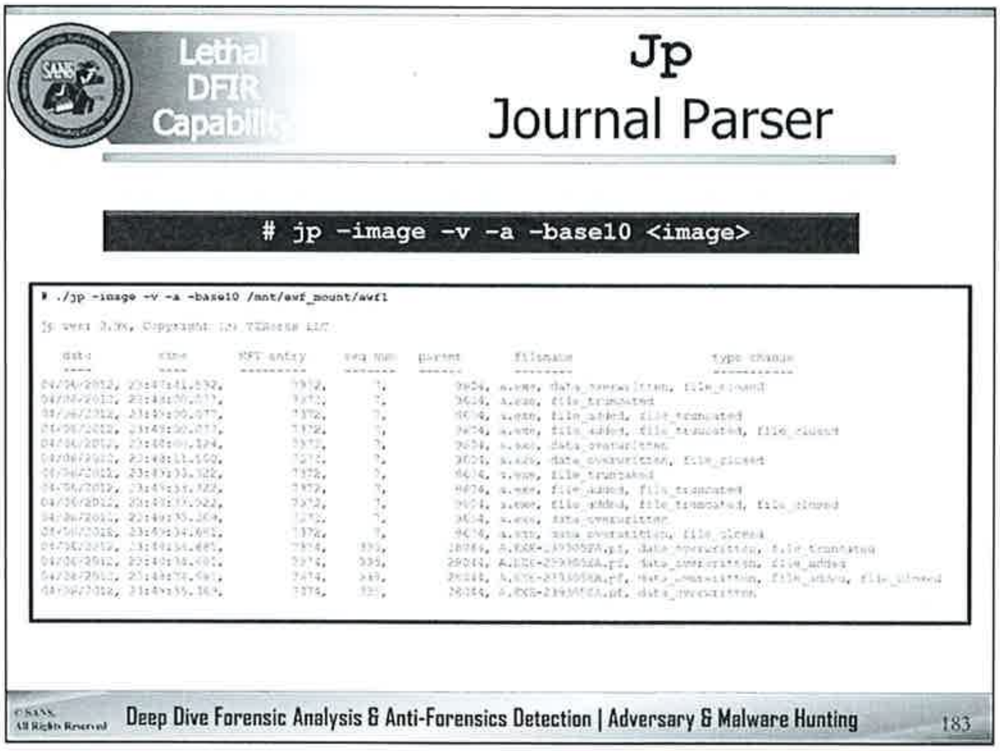
0x3C V Filename

Reason: The reason code is a bit field of flags that allow the system to track 21 different aspects of the file, to include but not limited to: content changed, directory was added to, ACL for file was changed, file name changed, attributes or times changed, compression state changed, encryption state changed, object ID changed.

This system file was introduced with NTFS version 3.0, and as such is not present on systems prior to XP. The Change journal feature was disabled by the default on XP and Vista, but is enabled by default on Windows 7. Any program can enable or disable this journal via an API call, therefore any program can flush this log and delete all of its data by disabling it.

http://msdn.microsoft.com/en-us/library/aa365722%28VS.85%29.aspx

http://www.forensickb.com/2008/09/enscript-to-parse-usnjrnl.html



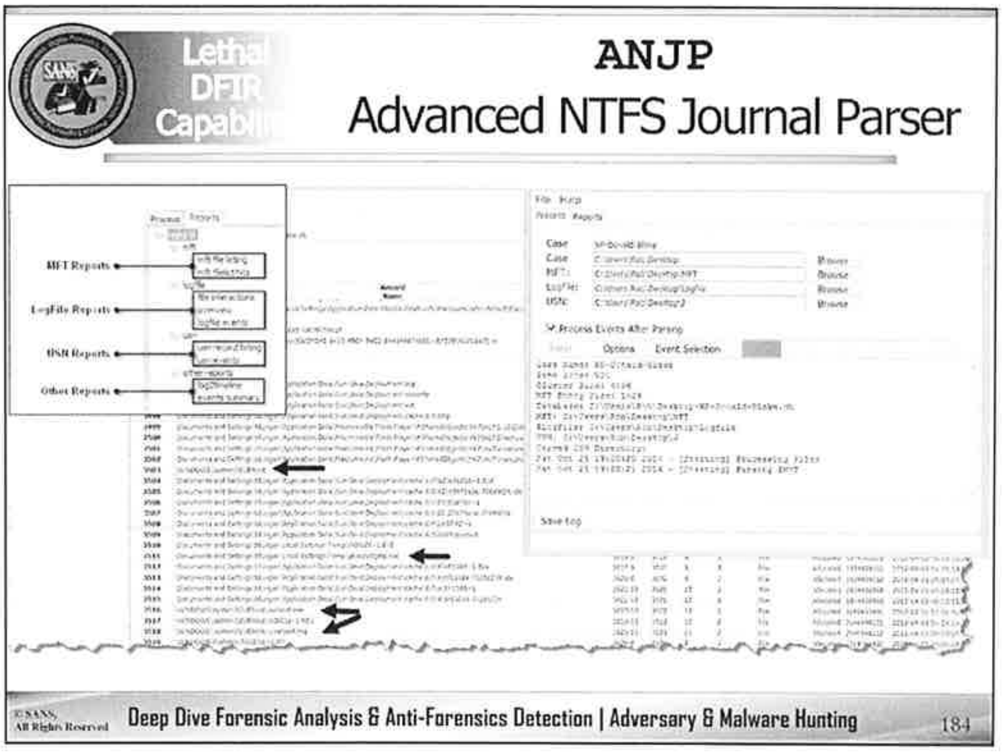
JP [1] is a small command line tool that does an outstanding job of parsing out the $UsnJrnl:$J index. This tool is cross platform, so there is a version for both the x86 and 64-bit versions of Windows, Linux, and Mac. It can parse the journal from a mounted file system, a raw disk image, or an extracted $UsnJrnl file's $J index. The $UsnJrnl file contains two named data streams – make sure you extract just the $J data.

Output is in the following order: date, time, MFT# sequence #, parent inode #, file name, and changes. The use the –v option adds the inode, sequence, and parent inode columns to the output, like in the example above. Also used in the example above is the –base10 option, which greatly increases the readability of the three columns added by the –v option.

```
usage:
 ./jp -file <extracted $UsnJrnl:$J file> [-v] [-a] [-xml]
 ./jp -image <disk image> [-offset <offset>] [-v] [-a] [-xml]
       -v = verbose output [includes MFT entry of file]
       -a = all records, not just those closed
       -xml = output in xml format
       -memory = will use minimal memory to run
       -base10 = output numbers in base10 vice hex

example of redirecting the output of change journal on c partition
 ./jp -partition c > output.txt
```

By greping for a file name, we can find all the entries that reference that name. Or by greping for an inode, like in the example above, we can target a specific file - useful if multiple files on volume have same name; also very useful in that we can follow file renames this way. As an interesting side note – notice how quickly the MFT record was reused once the file was deleted and how the sequence number helps keep the reference to the inode unique to specific file.

[1] **http://tzworks.net/prototype_page.php?proto_id=5**



ANJP provides a novel way of linking information contained in three important NTFS files that are responsible for maintaining the file system: The MFT, LogFile, and USN.

ANJP allows examiners to view file system activity stored within the system journals of an NTFS volume. Take a time-machine into the past to reveal the states of files and folders, including their location, size, name and more at specific points in the past. Zero in on historical file system activity such as file and folder creations, deletions, renames, moves, and much more using our event signatures. Retrieve file metadata that was lost due to overwriting or the use of anti-forensics techniques.

The LogFile, and USN can contain copious amounts of information. Manually sorting through hundreds of thousands of parsed records, and more importantly, understanding what the records mean could be resource intensive. ANJP does the work for you by searching for event signatures to reveal various kinds of file system activity such as file and folder creations, deletions, renames, moves, cd burns, LNK and prefetch deletions, ADS creations, virus infections, and much more.

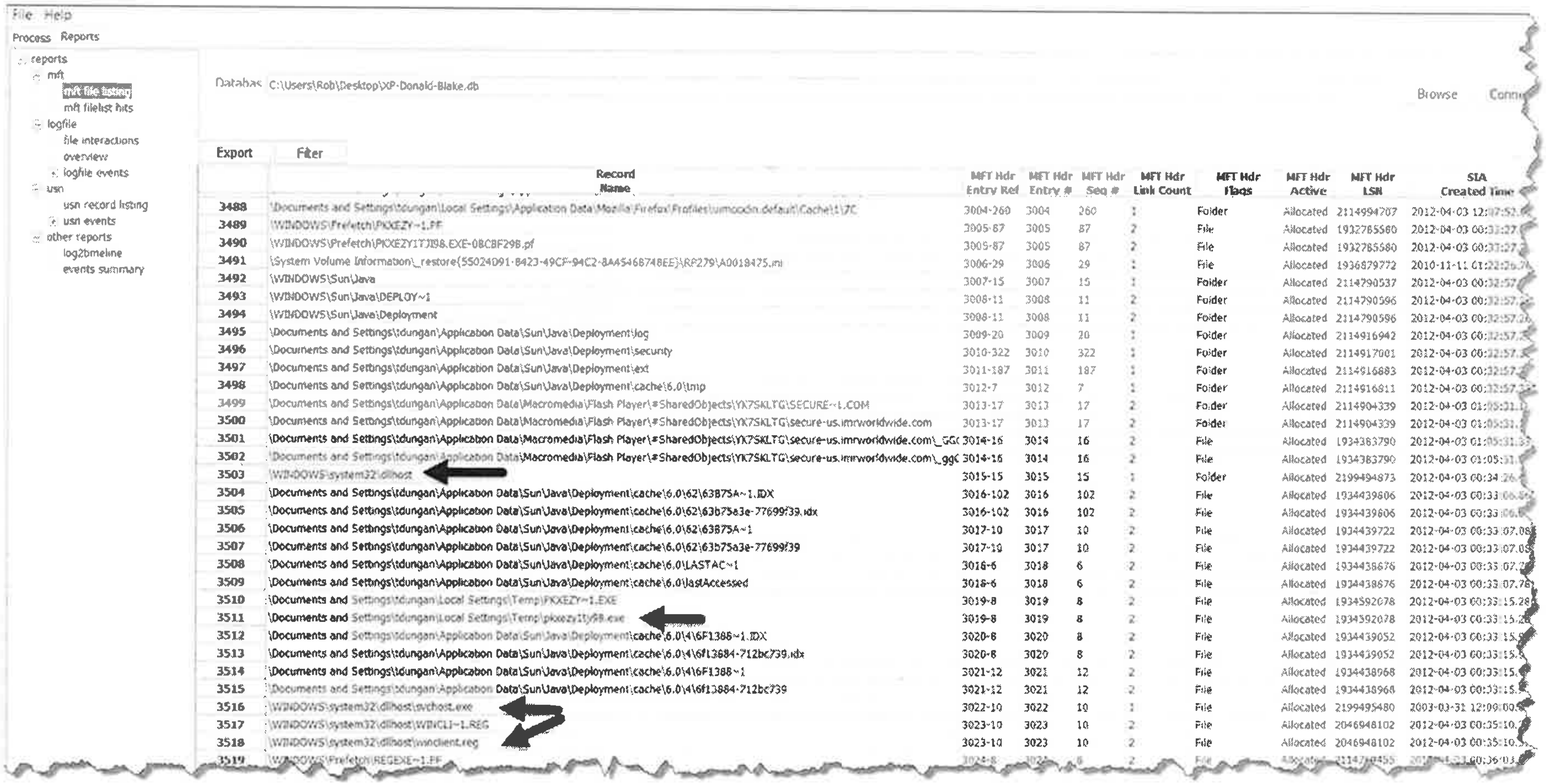
ANJP parses and links information allowing an examiner to track multiple changes that occurred to a specific file. For example, by filtering for a file's MTF reference number in an ANJP events report, you can track changes that occurred to the file such as, its creation, renaming, moving (within the same partition), deletion, and more.

By linking the LogFile and USN to the MFT, the fullpath (path and filename) for a given record within a LogFile or USN record can be enumerated. However, this linkage cannot be used to maintain fullpaths while parsing the entire LogFile or USN Journal. This is because as records are added to the LogFile and USN files can be deleted, created, and renamed, which potentially changes the path or filename for a given MFT entry. To overcome this, it is necessary to roll back the LogFile and USN to affect the correct fullpath for a given entry. Thus, parsing the LogFile and USN records from newest to oldest records, and applying changes to the fullpaths as files are deleted, created, and renamed results in the knowing exactly where a file was located and what its name was, when a change occurred.

Key Features of ANJP

- Recover historical metadata and full paths of files and folders throughout the LogFile and USN Journal
- View timestamp anomalies in the LogFile where timestamps are set back or zeroed out
- View timestamp anomalies in the MFT where timestamps are zeroed out or a Standard Information Attribute timestamp is less than that from the File Name Attribute timestamp
- Quickly identify items of interest with the use of MFT, LogFile, and USN signatures
- Create your own custom MFT file lists to search for matching filenames or full paths of entries parsed from the MFT
- Review records in various ANJP Report Views
- Create and apply custom filters to ANJP Report Views
- Export reports to XLSX or TXT

[1] https://www.gettriforce.com/product/anjp-free/



File Help

Process Reports

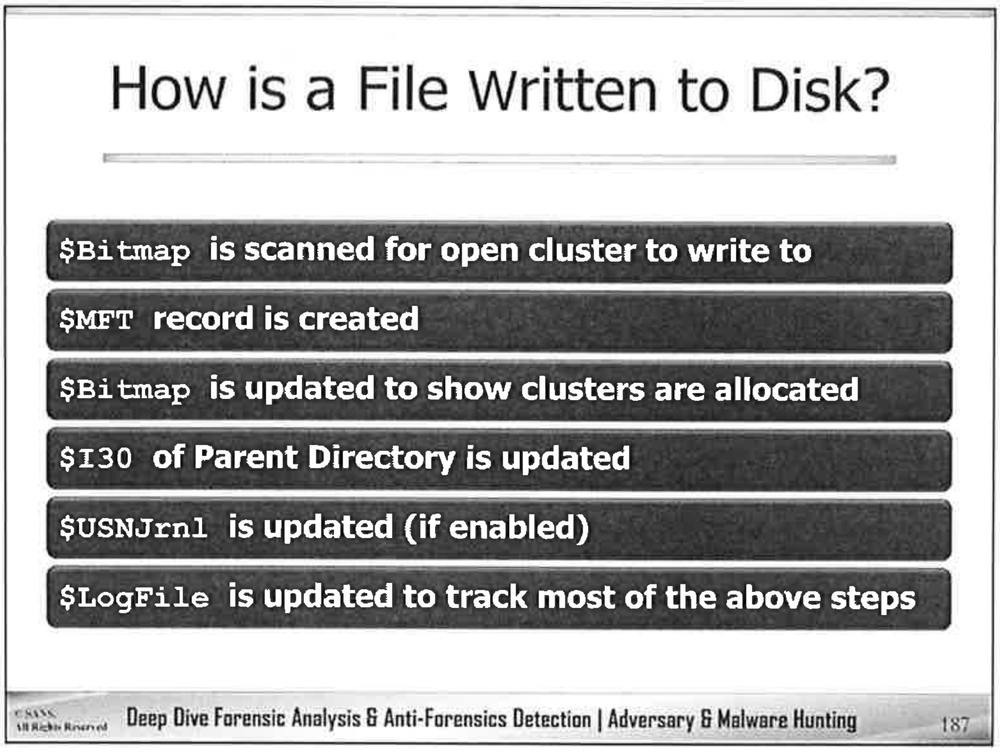
reports
- mft
  - mft file listing
  - mft filelist hits
- logfile
  - file interactions
  - overview
  - logfile events
- usn
  - usn record listing
  - usn events
- other reports
  - log2timeline
  - events summary

Databas C:\Users\Rob\Desktop\XP-Donald-Blake.db

Browse Conn

Export Filter

	Record Name	MFT Hdr Entry Ref	MFT Hdr Entry #	MFT Hdr Seq #	MFT Hdr Link Count	MFT Hdr Flags	MFT Hdr Active	MFT Hdr LSN	SIA Created Time
3488	\Documents and Settings\tdungan\Local Settings\Application Data\Mozilla\Firefox\Profiles\umoodn.default\Cache\1\7C	3004-260	3004	260	1	Folder	Allocated	2114994707	2012-04-03 12:07:52
3489	\WINDOWS\Prefetch\PKXEZY~1.PF	3005-87	3005	87	2	File	Allocated	1932785580	2012-04-03 00:33:27
3490	\WINDOWS\Prefetch\PKXEZY1TJI98.EXE-0BCBF29B.pf	3005-87	3005	87	2	File	Allocated	1932785580	2012-04-03 00:33:27
3491	\System Volume Information_restore{55024D91-8423-49CF-94C2-8A4546874BEE}\RP279\A0018475.ini	3006-29	3006	29	1	File	Allocated	1936879772	2010-11-11 01:22:26.7
3492	\WINDOWS\Sun\Java	3007-15	3007	15	1	Folder	Allocated	2114790537	2012-04-03 00:32:57
3493	\WINDOWS\Sun\Java\DEPLOY~1	3008-11	3008	11	2	Folder	Allocated	2114790596	2012-04-03 00:32:57
3494	\WINDOWS\Sun\Java\Deployment	3008-11	3008	11	2	Folder	Allocated	2114790596	2012-04-03 00:32:57
3495	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\log	3009-20	3009	20	1	Folder	Allocated	2114916942	2012-04-03 00:32:57
3496	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\security	3010-322	3010	322	1	Folder	Allocated	2114917001	2012-04-03 00:32:57
3497	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\ext	3011-187	3011	187	1	Folder	Allocated	2114916883	2012-04-03 00:32:57
3498	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\tmp	3012-7	3012	7	1	Folder	Allocated	2114916811	2012-04-03 00:32:57
3499	\Documents and Settings\tdungan\Application Data\Macromedia\Flash Player\#SharedObjects\YK7SKLTG\SECURE~1.COM	3013-17	3013	17	2	Folder	Allocated	2114904339	2012-04-03 01:05:31
3500	\Documents and Settings\tdungan\Application Data\Macromedia\Flash Player\#SharedObjects\YK7SKLTG\secure-us.imrworldwide.com	3013-17	3013	17	2	Folder	Allocated	2114904339	2012-04-03 01:05:31
3501	\Documents and Settings\tdungan\Application Data\Macromedia\Flash Player\#SharedObjects\YK7SKLTG\secure-us.imrworldwide.com_GG(	3014-16	3014	16	2	File	Allocated	1934383790	2012-04-03 01:05:31
3502	\Documents and Settings\tdungan\Application Data\Macromedia\Flash Player\#SharedObjects\YK7SKLTG\secure-us.imrworldwide.com_gg(	3014-16	3014	16	2	File	Allocated	1934383790	2012-04-03 01:05:31
3503	\WINDOWS\system32\dllhost	3015-15	3015	15	1	Folder	Allocated	2199494873	2012-04-03 00:34:26
3504	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\62\63B75A~1.IDX	3016-102	3016	102	2	File	Allocated	1934439606	2012-04-03 00:33:06
3505	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\62\63b75a3e-77699f39.idx	3016-102	3016	102	2	File	Allocated	1934439606	2012-04-03 00:33:06
3506	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\62\63B75A~1	3017-10	3017	10	2	File	Allocated	1934439722	2012-04-03 00:33:07.08
3507	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\62\63b75a3e-77699f39	3017-10	3017	10	2	File	Allocated	1934439722	2012-04-03 00:33:07.08
3508	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\LASTAC~1	3018-6	3018	6	2	File	Allocated	1934438676	2012-04-03 00:33:07.7
3509	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\lastAccessed	3018-6	3018	6	2	File	Allocated	1934438676	2012-04-03 00:33:07.78
3510	\Documents and Settings\tdungan\Local Settings\Temp\PKXEZY~1.EXE	3019-8	3019	8	2	File	Allocated	1934592078	2012-04-03 00:33:15.28
3511	\Documents and Settings\tdungan\Local Settings\Temp\pkxezy1tj98.exe	3019-8	3019	8	2	File	Allocated	1934592078	2012-04-03 00:33:15.2
3512	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\4\6F1388~1.IDX	3020-8	3020	8	2	File	Allocated	1934439052	2012-04-03 00:33:15.
3513	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\4\6f13884-712bc739.idx	3020-8	3020	8	2	File	Allocated	1934439052	2012-04-03 00:33:15.
3514	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\4\6F1388~1	3021-12	3021	12	2	File	Allocated	1934438968	2012-04-03 00:33:15.
3515	\Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\4\6f13884-712bc739	3021-12	3021	12	2	File	Allocated	1934438968	2012-04-03 00:33:15.
3516	\WINDOWS\system32\dllhost\svchost.exe	3022-10	3022	10	1	File	Allocated	2199495480	2003-03-31 12:09:00.
3517	\WINDOWS\system32\dllhost\WINCLI~1.REG	3023-10	3023	10	2	File	Allocated	2046948102	2012-04-03 00:35:10.
3518	\WINDOWS\system32\dllhost\winclient.reg	3023-10	3023	10	2	File	Allocated	2046948102	2012-04-03 00:35:10.
3519	\WINDOWS\Prefetch\REGEXE~1.PF	3024-8	3024	8	2	File	Allocated	[illegible]	[illegible] 00:36:03



The above steps are in no particular order. And, it is probably not even all of the steps that actually occur.

To improve performance, all of these writes will take place in memory in a cache maintained by the file system drivers. The cache will periodically get flushed and changes will be written to disk. This is why it is bad to unplug a drive without unmounting it, e.g. removing a USB flash drive or external drive, or abruptly cutting power to the system. If the drive is removed without flushing the cache, some of the above steps will not have been written to the disk yet. If the system detects that not all steps are done, upon next mount the system will roll back any unfinished transactions, causing loss of data. Generally, completing a transaction triggers a flush of the cache so that everything gets committed to disk, but sometimes, especially if there is a lot of a writes to the volume coming in at the same time, a queue could build up and it could take a little while before it is all committed to disk.

NTFS: What Data Still Exists Upon File Deletion?

FILENAME LAYER

- `$File_Name` attribute is preserved until MFT record is reused
- `$I30` index entry in parent directory may be preserved

METADATA LAYER

- Only one bit in MFT record will change, so all file metadata will remain until record is reused
- `$LogFile`, `$USNJrnl`, and other system logs may still reference file

DATA LAYER

- Data clusters will be marked as unallocated in `$Bitmap` but data will be preserved until clusters are reused
- Slack Space will exist

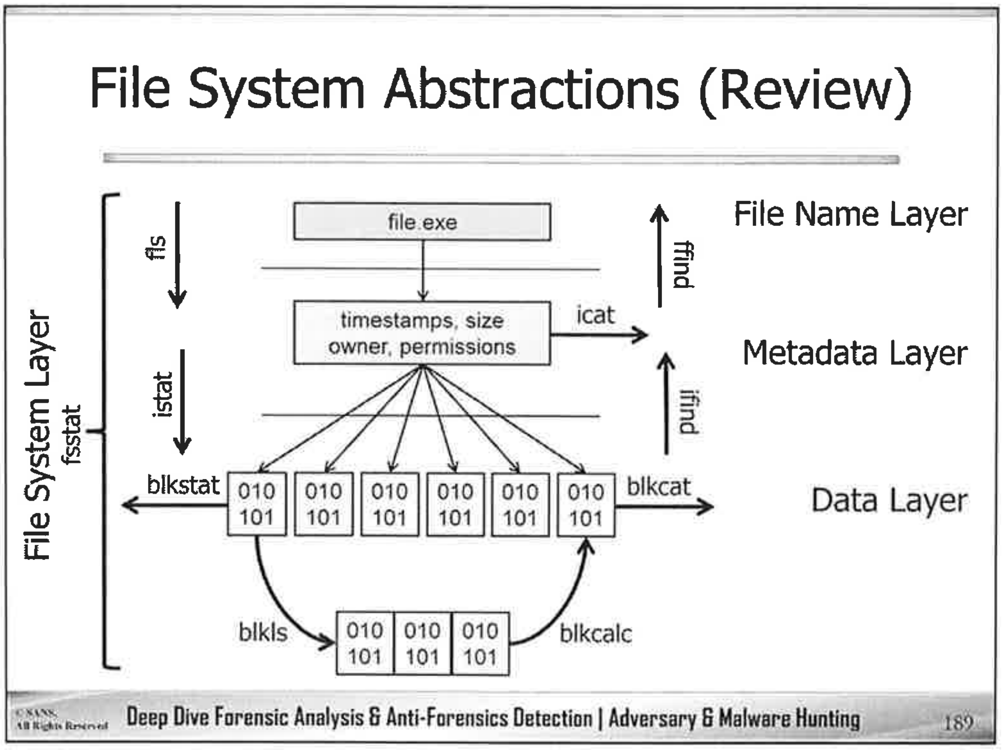
When a file is deleted, most of the steps from the previous slide are reversed.

- MFT record is **marked as available** – not immediately overwritten, so may exist completely intact
- $Bitmap file marks associated clusters as available – **Clusters themselves are never touched**
- Parent Directory's index is rewritten to remove references to this file **or** to mark the entry as available
- $LogFile is updated to reflect transaction occurred
- $USNJrnl is updated (if enabled) to reflect file's deletion
- $Secure, $ObjID, and $Quota are updated (if applicable to the file)

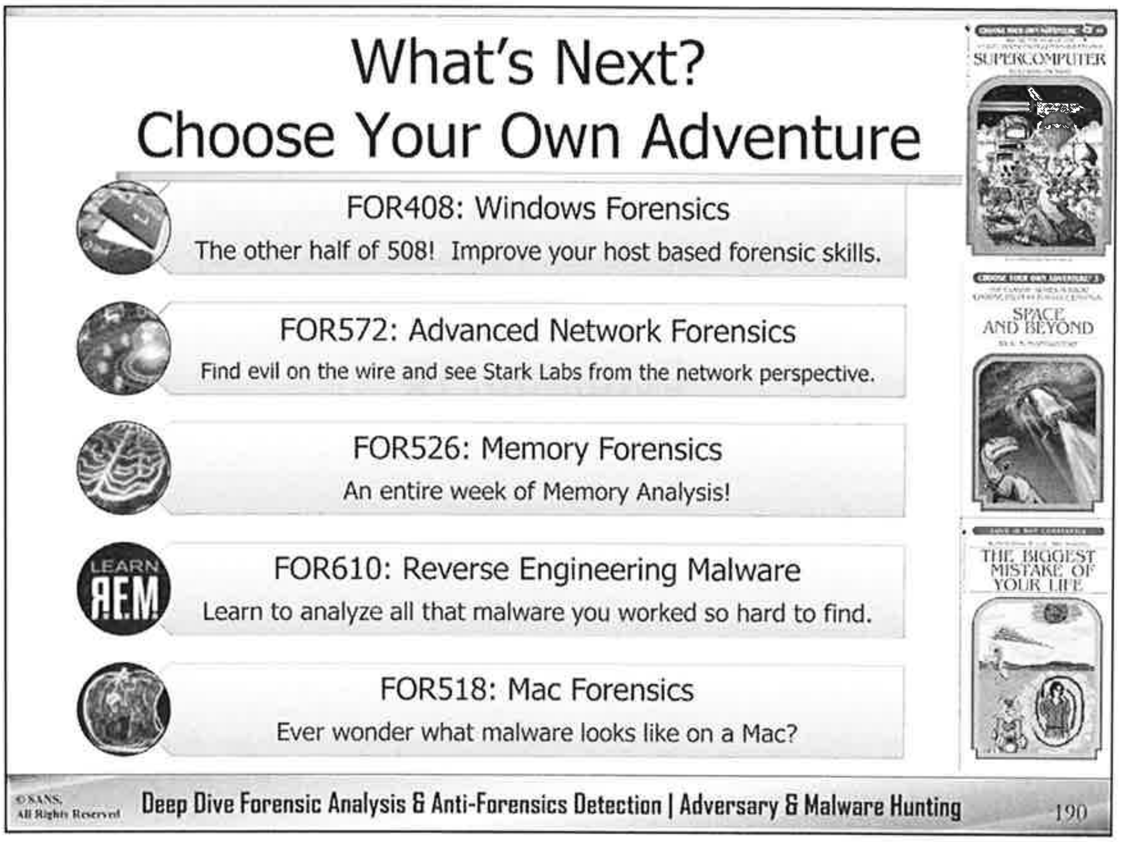
Just as with FAT and other file systems, deleting a file deallocates the clusters by updating the $Bitmap file, but leaves the data in place for us forensics examiners to look at later. The MFT Record has the "in use" flag (a single bit) flipped in the Record header to mark that the record is no longer in use. MFT Records will get reused, causing the deleted file's metadata to be overwritten. Sometimes this happens right away, while other records remain for considerable time – there is too little consistency to be able to predict when the system will reuse a record, and thus how long to expect to be able to find metadata for a deleted file.

Depending on several variables regarding how large and active the file's parent directory is, the file's entry in the $I30 index could either be deleted causing the entire index to be rewritten with all the entries after that one being shifted up, or the deleted entry may stay in place and marked as not allocated so the system will step over it when reading the index.

The $Secure, $ObjID, and $Quota indexes are modified to remove or update references to the file. The $Logfile will gain a series of entries related to the deletion transaction. The $UsnJrnl will gain an entry, if enabled, telling us the file was deleted.



This slide recaps the various file system abstraction categories, and how the various Sleuthkit tools graphically relate to each other.



This page intentionally left blank.

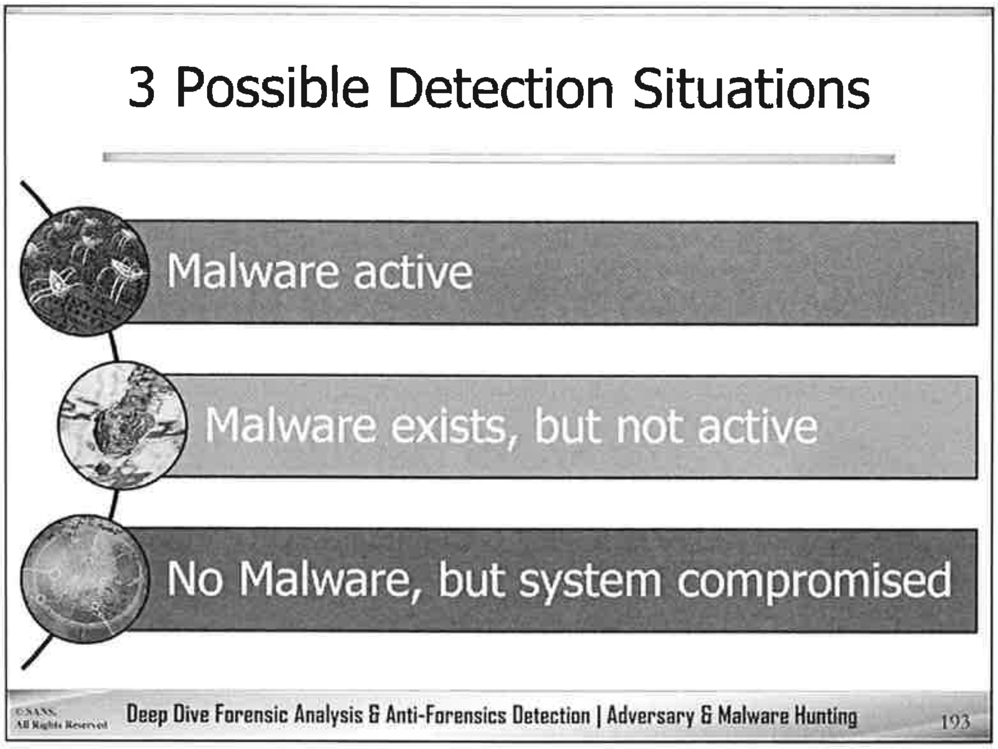
SANS Digital Forensics and Incident Response
CURRICULUM

Exercise 18

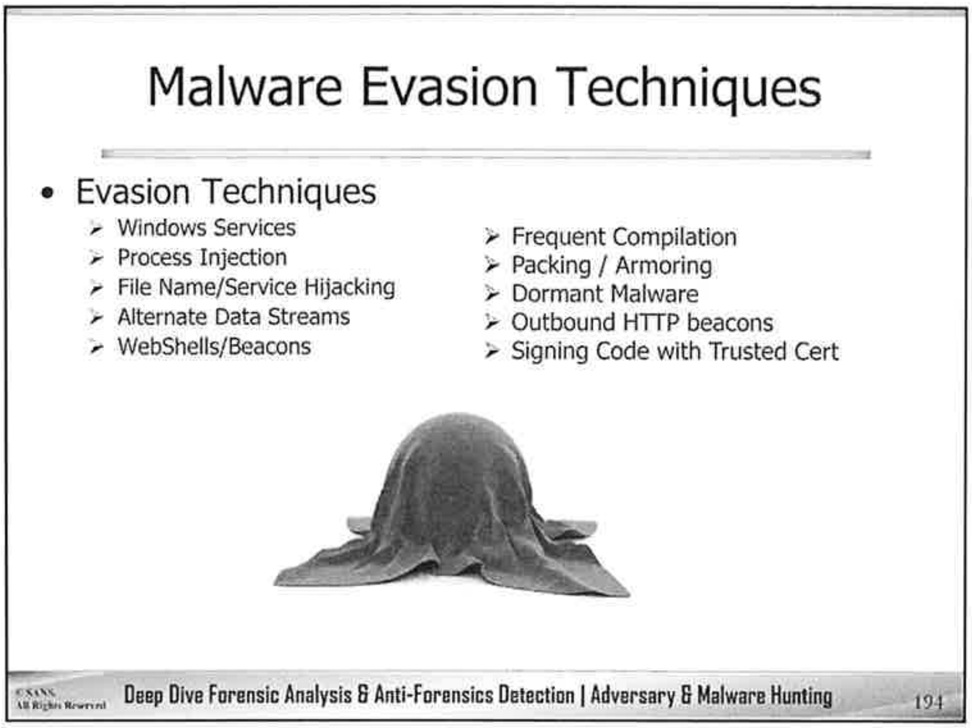
NTFS Filesystem Analysis

This page intentionally left blank.

Author: Rob Lee
rlee@sans.org
http://twitter.com/robtlee
http://twitter.com/sansforensics



When hunting for a compromise, it is actually easiest if there is active malware on the system. In most cases, it gives you more places to look. Malware that is not active, but is dormant is harder to detect as we lose the ability to detect the malware in memory. Malware that is dormant could launch through a specific persistence mechanism like a Word Add-On or via a scheduled task instead of at boot. Finally, the last situation to consider is the system is compromised and malware doesn't currently exist. In this last situation, it is actually the hardest of the situations.

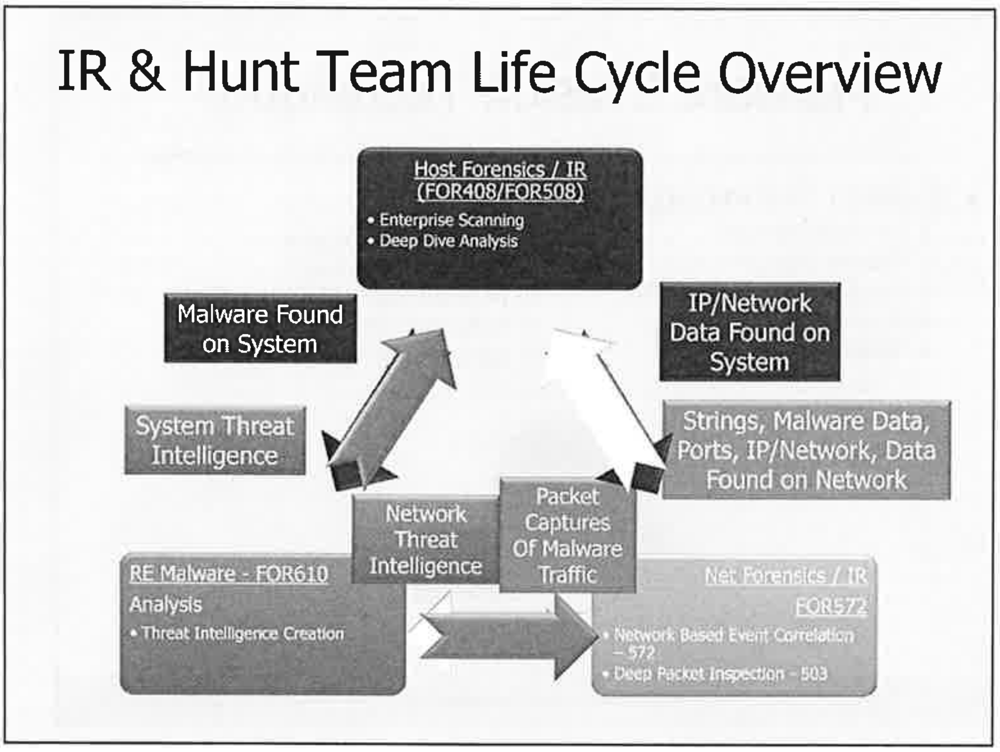


First, look at how malware is typically found in cases today. Most malware will simply hide in plain sight. No need for fancy rootkits for today's advanced hackers. They know that rootkit technology is often detectable and often creates system instability. As a result, malware will simply try to hide among the thousands of files and directories that exist on a standard windows system.

Most analysts who become decent forensicators have no trouble finding where malware might exist after they gain years of experience. It isn't a difficult process to follow what they know, but it is half process and half art. Malware traditionally will probably create a new service, using the scheduler "at" command that makes the malware hide in plain sight among other normal services. However, I doubt the service name will be called "EVIL MALWARE BRU-HAHAHA SERVICE", more likely it will also look like a normal service. In some cases, but more rare, the malware will replace an existing service such as wireless zero configuration service. The third most popular method that malware will maintain persistence is by registering as an auto-run process in the registry. And finally, process injection is also used, but not as frequently.

Some other traits associated with malware include that malware today is frequently recompiled; use alternate data streams, frequently packed and armored to prevent A/V detection and anti-reversing. It is also very common to find dormant malware on a machine. Either to be activated via a remote scheduler from another machine in your network or that the malware was improperly deleted/wiped when the adversary last used the system.

Let's take a look at how malware is typically found in cases today. Malware often employs tactics that are categorized as "Hiding in Plain Sight". What does this mean? Adversaries know full well that the more hiding techniques employ the more likely a Host Based Intrusion Detection System or Anti Virus program will detect them – this includes rootkit techniques – the more manipulation of the Windows API, the more likely one is to



In order to use security intelligence effectively during the 6 step IR process. You need to collect your data to be processed to look for atomic, computed, and behavioral indicators. To accomplish this, you need to set up your team correctly so that your IR team is focused on intelligence creation as much as possible during the identification and containment phase of IR.

Digital forensics and IR is the process to analyze systems (host and network data) in order to properly identify all systems compromised and the ability to learn exactly what remediation steps will need to be taken in order to stop the incident. As a part of forensics, being able to properly collect intelligence of the adversary will depend greatly on your team's ability to analyze the remnants of the activities of the intruder and then use that data as a way to "catch" them on other systems.

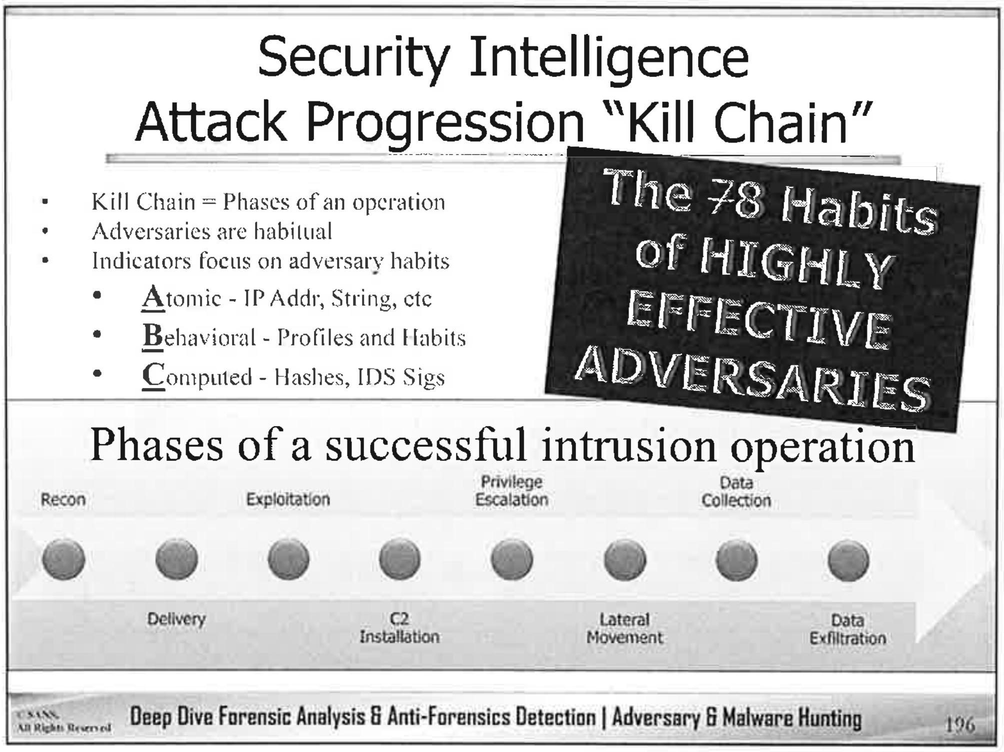
As a result, your team is likely to be made up of host, network, and reverse engineers that will be working side by side to help identify new systems compromised, create new threat intelligence data, and finally use that data to identify new systems and engineer additional defenses that can be used to help stabilize the current incident.

The challenge today is that our adversaries might not simply "go away." It is likely they will come back to your network again and deploy more advanced techniques in order to maintain persistence.

In order to enact the kill chain successfully, each IR team should have a team with a variety of skills. Each group helps support the other groups.

- 1 Team lead
- 1-2 System and host forensicators
- 1-2 Network forensicators
- 1 RE Malware Specialists

Finally, the team should have an enterprise scanning capability looking for both host and network based signatures discovered during the intrusion. This will help enable the ability to stop beach heads before they are able to laterally move or exfiltrate your data from the network.



Source with permission from Mike Cloppert originally published on the SANS Computer Forensics Blog: http://digital-forensics.sans.org/blog/2009/10/14/security-intelligence-attacking-the-kill-chain/

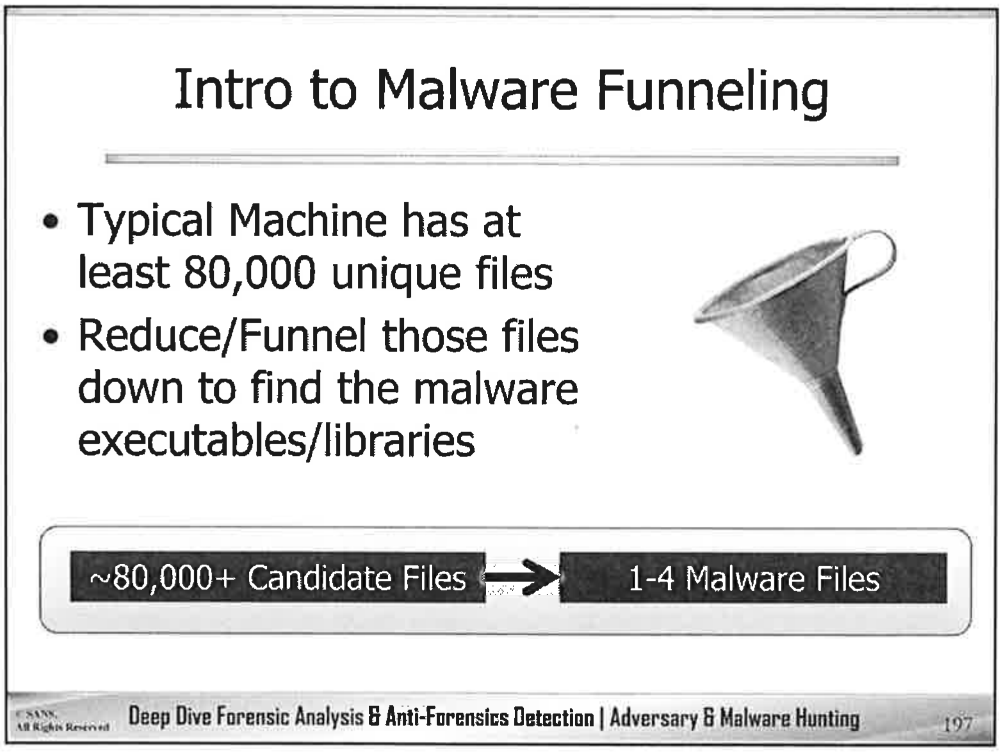
Now we will introduce the attack progression (aka "kill chain") and briefly describe its intersection with indicators. The next segment will go into more detail about how to use the attack progression model for more effective analysis and defense, including a few contrived examples based on real attacks.

On Indicators

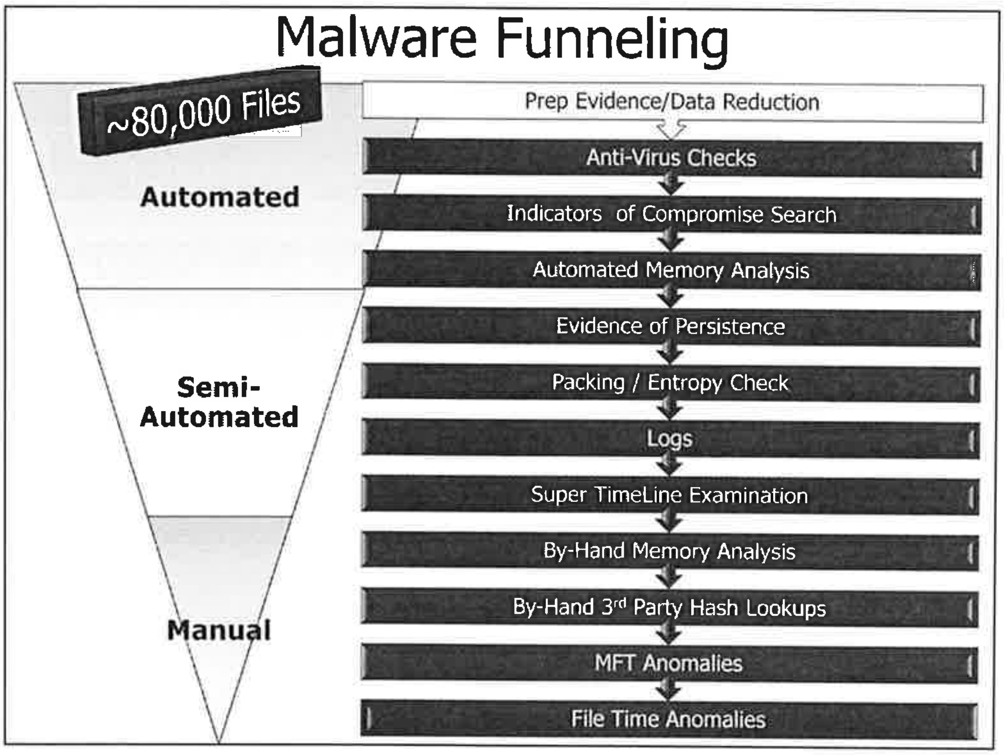
Just like you or I, adversaries have various computer resources at their disposal. They have favorite computers, applications, techniques, websites, etc. It is these fundamentally human tendencies and technical limitations that we exploit by collecting information on our adversaries. No person acts truly random, and no person has truly infinite resources at their disposal. Thus, it behooves us in CND to record, track, and group information on our sophisticated adversaries to develop profiles. With these profiles, we can draw inferences, and with those inferences, we can be more adaptive and effectively defend our data. After all, that's what intelligence-driven response is all about: defending data that sophisticated adversaries want. It's not about the computers. It's not about the networks. It's about the data. We have it, and they want it.

Indicators can be classified a number of ways. Over the years, I and my colleagues have wrestled with the most effective way to break them down. Currently, I am of the mind that indicators fall into one of three types: atomic, computed, and behavioral (or TTP's).

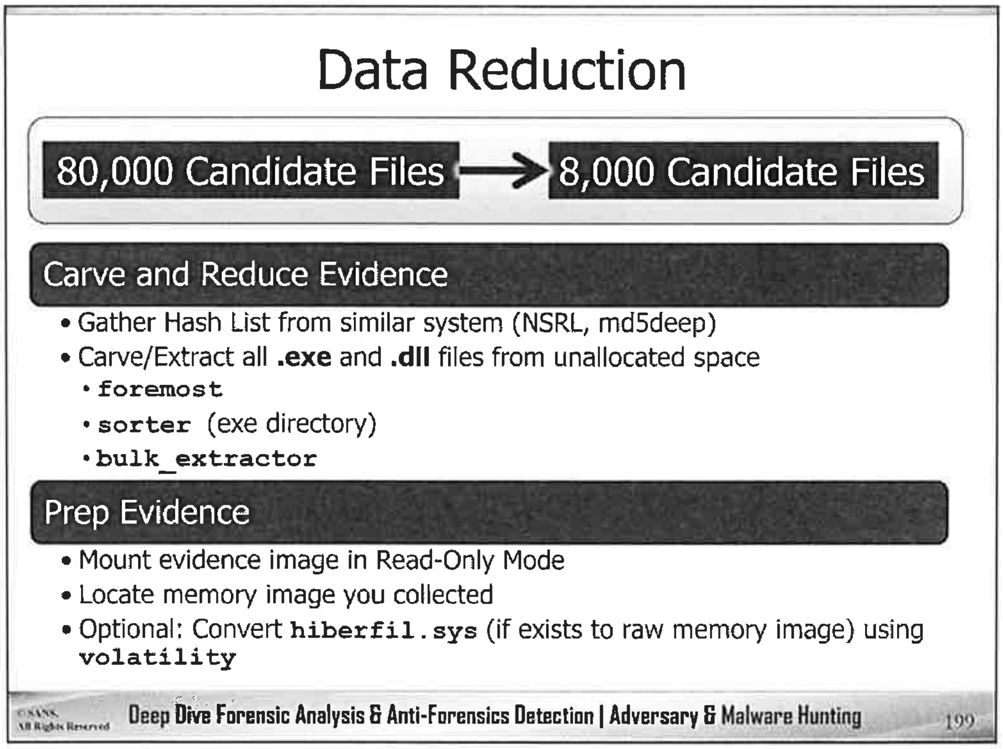
Atomic indicators are pieces of data that are indicators of adversary activity on their own. Examples include IP addresses, e-mail addresses, a static string in a Covert Command-and-control (C2) channel, or fully-qualified



Our goal when we first begin our journey to find evil on our machine is to narrow down all the files on the machine to the one or two files that will be our malware. This is not an easy process, but using the techniques in this section you will learn how to narrow the 80,000 files on a typical machine down to the 1-4 files that is our malware. This process of Malware Funneling is key to your quick and efficient analysis of compromised hosts and will involve most of the skills you have built up across both FOR408 and FOR508. Many of these sections will be considered "Review" but it is how we use it to help us narrow the candidates down further that will be useful in this specific type of case.



This page intentionally left blank.



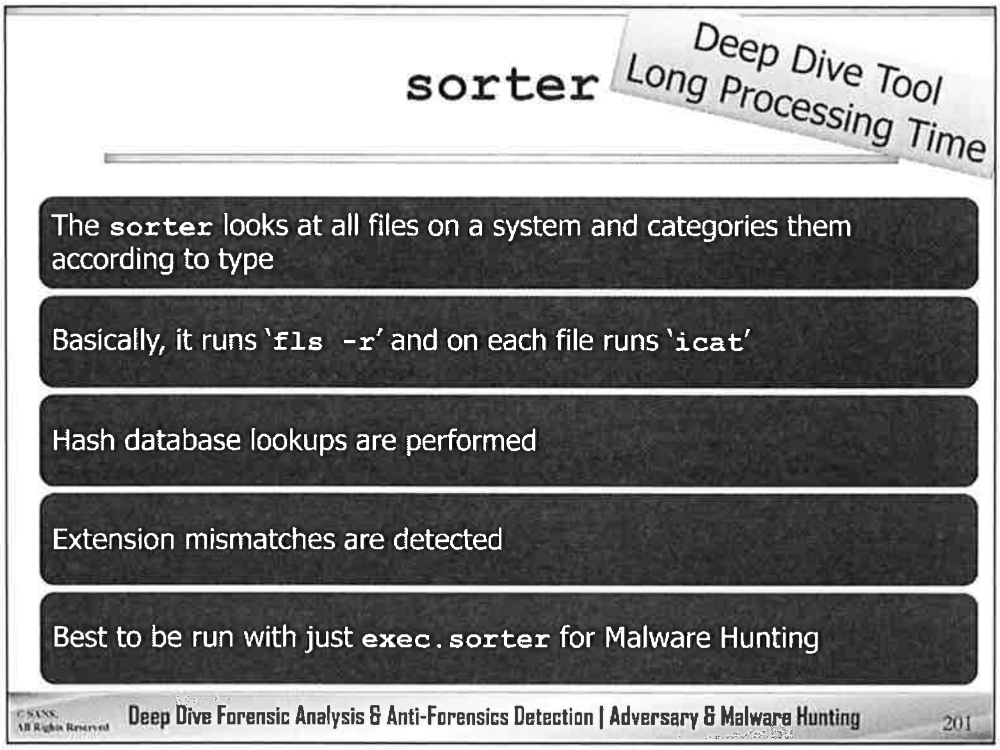
When you begin to look for malware on a machine you first must have already carved out of the unallocated space as many intact .exe files as possible. You also should consider running sorter on the image to collect any files from the metadata layer that are also still recoverable.

Generally, once you have a mounted image, carved files, and sorted files, you are ready to begin.

SANS Digital Forensics and Incident Response
CURRICULUM

Data Reduction: Hash Comparisons

This page intentionally left blank.



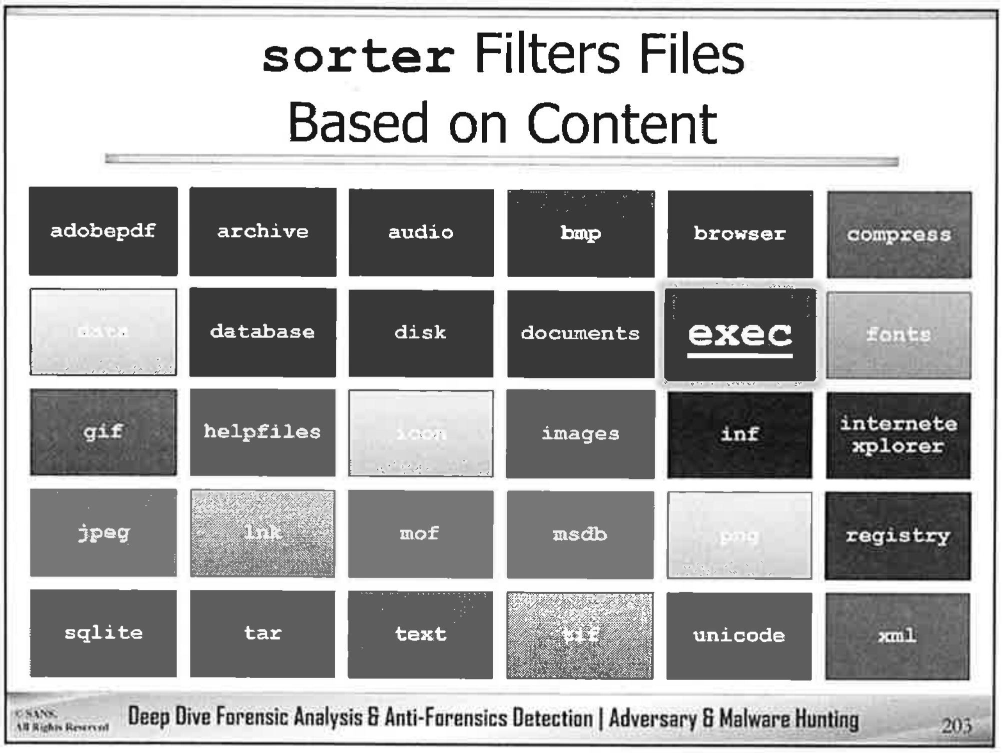
The sorter tool is a perl script that combines many small tools from The Sleuth Kit. It runs the 'fls' tool to get a list of full file paths and then runs 'icat' on each. That data is sent to the 'file' command that identifies what file type it is. It then either saves the data to a directory or just makes a note of it. The tool can also lookup into hash databases to see if the file is known or not. Typically if you have a basic baseline image to compare it to, excluding will result in a good percentage of known files being examined deeply which is very useful.

Sorter takes the disk image as input and runs FLS against every file on the system. Not only will it run fls, it will run fls against every file in the evidence image including deleted files. Then, it will cat out the contents of every file on the system and pipe it to the program file categorizing file contents and saving those files to specific category directories.

Custom configuration file for /usr/share/tsk/sorter/exec.sort

```
#
# config file for Sleuth Kit sorter
#
# Windows Platform Executables
##########################################################################
category    exec            MS\-DOS executable
ext         exe,dll,com     MS\-DOS executable
ext         ocx,sys,tlb     MS\-DOS executable
ext         drv,cpl,scr     MS\-DOS executable
```

ext	ax	MS\-DOS executable
ext	386,acm,flt	MS\-DOS executable
ext	fon,lrc,vxd	MS\-DOS executable
ext	x32	MS\-DOS executable
category	exec	executable MS\-DOS
ext	exe	MZ executable MS\-DOS
ext	com	MZ executable MS\-DOS
category	exec	PE executable MS Windows
ext	exe,dll,com	PE executable MS Windows
ext	ocx,sys,acm	PE executable MS Windows
ext	tlb,drv,scr	PE executable MS Windows
ext	cpl,ax,vdx	PE executable MS Windows
ext	fon,rll,tsp	PE executable MS Windows
category	exec	PE32 executable
ext	exe,dll,com	PE32 executable
ext	ocx,sys,acm	PE32 executable
ext	tlb,drv,scr	PE32 executable
ext	cpl,ax,vdx	PE32 executable
ext	fon,rll,tsp	PE32 executable
category	exec	NE executable MS Windows
ext	exe,dll,com	NE executable MS Windows
ext	ocx,sys,acm	NE executable MS Windows
ext	tlb,drv,scr	NE executable MS Windows
ext	cpl,ax,vxd	NE executable MS Windows
ext	fon,tsp	NE executable MS Windows
category	exec	relocatable
ext	dll	relocatable
category	exec	batch file
ext	bat	batch file
ext	bat	ASCII text
ext	bat	ASCII English text
ext	nt	DOS batch file
ext	cmd	DOS batch file
# source code		
category	exec	MSVC program database
ext	pdb	MSVC program database
category	exec	\sscript
# Adobe		
category	AdobePDF	PDF document
ext	pdf	PDF document
# Crypto		



Sorter is able to carve files out based on the following data types based on the CONTENT of the file, not the extension. It will then create a directory and a .html file that will map the file back to the original filename. This is useful because it will sort files (existing and deleted files) regardless of their status. This will make finding malware or adobe PDF files much easier.

The sorter on SIFT has different configuration files than the default sleuthkit distribution. We expanded the number of file type categories analyzed by sorter.

In some cases, I actually run sorter against a directory that I extracted onto a USB using pffexport (the pst extraction tool) to sort the attachments sent in an e-mail. What this will allow you to do is to find all the PDFs in an archive quicker than simply trying to extract each one at a time. I would then image the USB key that contains just the pffexport of the e-mail, and save to a disk image. I would then use sorter against that disk image or device that contained the pffexport directory structure. That way I can quickly examine ALL the pdf files and office document files in a single directory without much work.

Steps from above:

1. pffexport outlook.pst to a thumbdrive
2. Image USB to an image file
3. Run sorter on image file to categorize attachments

Sorter initially might not seem that powerful a tool, but when you begin to use it in your casework, it will really change your perspective. The only downside to the tool is the length of time it needs to run. Plan for at least a couple of hours for an executable file sort.

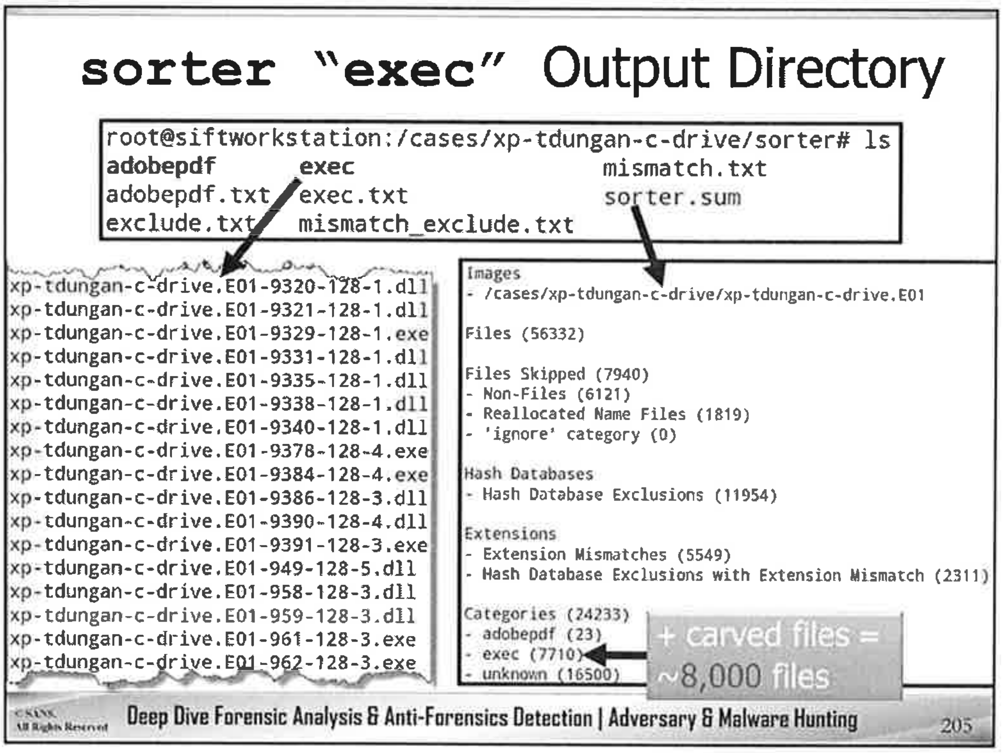

```
# hfind -i md5sum /cases/xp-tdungan-cdrive/precooked/hashes/WinXPSP3x86.txt
# sorter -U -s -m C: -x /cases/xp-tdungan-
cdrive/precooked/hashes/WinXPSP3x86.txt -d sorter -C
/usr/share/tsk/sorter/exec.sort /cases/xp-tdungan-c-drive/xp-tdungan-c-
drive.E01
```

The flags for sorter deal with how much work it will have to do. By default it does both extension mismatch and category indexing. They can be turned off at compile time if you wish.

By default, only an entry into a file is created (based on type). Using the 's' option, the data will be saved to a file in a directory based on type. The '-d' argument shows where the root directory for the save will be.

If the system is compromised, we can use the '-x' or the '-n' option of sorter and all files that are found in the hash database will be excluded. Therefore, we will not see them in the sorter output and we can focus on files that may have been created by the attack. We can also use the '–a' option to alert on specific files that are found.

Once you have your databases whether it is known good or known bad, which you will need to do your indexing and your collection for both known good and known evil. You will then use your sorter tool to be able to go through and alert on the hash list for known evil files, exclude hash list unknown good files, and the NSRL database. You have to probably go manually and carve out any of the bad files that you think are hovering around in there and potentially add in back up into the known bad files one. This is why this requires a little bit more manual process.



This is the sorter output directory once sorter has completed its run across an image collecting only executable files. You can open up any of the html files in Firefox to examine the files that were recovered and their original location and MFT entry.

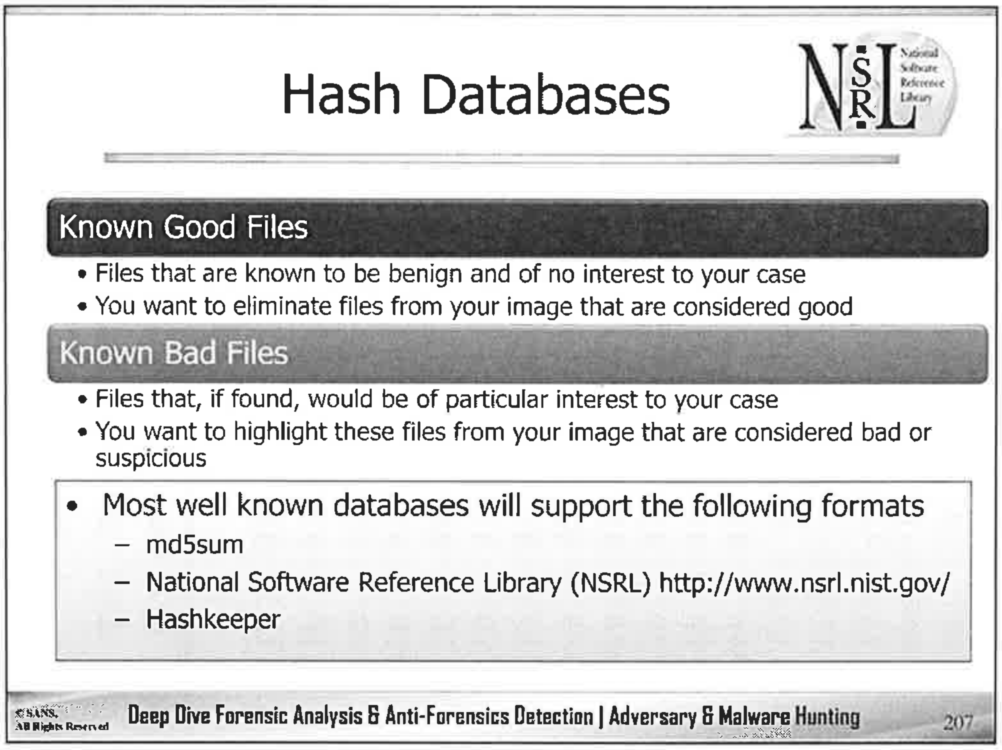
The exclude.txt file will contain a list of all files that the known_goods hash list helped exclude from examination.

```
root@siftworkstation:/cases/xp-tdungan-c-drive/sorter# ls
adobepdf      exec                        mismatch.txt
adobepdf.txt  exec.txt                    sorter.sum
exclude.txt   mismatch_exclude.txt
```

```
xp-tdungan-c-drive.E01-9320-128-1.dll
xp-tdungan-c-drive.E01-9321-128-1.dll
xp-tdungan-c-drive.E01-9329-128-1.exe
xp-tdungan-c-drive.E01-9331-128-1.dll
xp-tdungan-c-drive.E01-9335-128-1.dll
xp-tdungan-c-drive.E01-9338-128-1.dll
xp-tdungan-c-drive.E01-9340-128-1.dll
xp-tdungan-c-drive.E01-9378-128-4.exe
xp-tdungan-c-drive.E01-9384-128-4.exe
xp-tdungan-c-drive.E01-9386-128-3.dll
xp-tdungan-c-drive.E01-9390-128-4.dll
xp-tdungan-c-drive.E01-9391-128-3.exe
xp-tdungan-c-drive.E01-949-128-5.dll
xp-tdungan-c-drive.E01-958-128-3.dll
xp-tdungan-c-drive.E01-959-128-3.dll
xp-tdungan-c-drive.E01-961-128-3.exe
xp-tdungan-c-drive.E01-962-128-3.exe
```

```
Images
- /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01

Files (56332)

Files Skipped (7940)
- Non-Files (6121)
- Reallocated Name Files (1819)
- 'ignore' category (0)

Hash Databases
- Hash Database Exclusions (11954)

Extensions
- Extension Mismatches (5549)
- Hash Database Exclusions with Extension Mismatch (2311)

Categories (24233)
- adobepdf (23)
- exec (7710)
- unknown (16500)
```

+ carved files =
~8,000 files



Having a database of hash values can save a lot of time. It will allow you to identify files that are known to be bad and ones that are known to be good. The hfind tool in TSK will allow you to quickly lookup files in such a database.

What we need to do in order for us to be able to do a hash comparison, you have to have a hash database. The hash database is at least a precooked database of either known goods and known evil files. Most of the databases will be a MD5 hash database or SHA1 hash databases for both known good and bad hashes.

The known good files or the files that are known to be benign and of no interest to your case. You don't care if they exist on your machine. You want to eliminate the files from your images that are considered good.

The known bad files, the known evil. If you find this file, this would be particular interest to you on this machine. You want to highlight these files from the image that are considered bad or suspicious.

While the National Software Reference Library is an arm of NIST (National Institute Science & Technology) that they have already done a little bit of this work for you. They have already created a known hash list of all the major operating systems and known applications. Now, the annoying thing is about the National Software Reference Library is that it is far from being complete. However, all things being considered, it is one of the best freely available databases available for download.

The database is only four cdroms that you could download from the website at: http://www.nsrl.nist.gov

Note: If using the hash database features of sorter, the database must first be indexed by using the tool hfind.

Create Hash List: `md5deep`

- Written by Jesse Kornblum
- How do we create a list of known_goods or known_bads? **`md5deep`**
- Tools designed to recursively go through a file system to calculate
  - MD5, SHA-1, SHA-256, Tiger, or Whirlpool Hashes
  - **`md5deep, sha1deep, sha256deep, tigerdeep,`** or **`whirlpooldeep`**
- Usage:

```
# md5deep -r starting_directory > hash_outputfile.txt
```

- Options

  `-r` = recursive mode

  Example:

```
# ./md5deep -r / > known_good_files.txt
```


MD5DEEP computes the MD5 message digest for any number of files while optionally recursively digging through the directory structure. The way you create your hash is, is by using a tool called md5deep. The way it works is you run md5deep in recursive mode, add a starting directory, and you redirect that down into a hash output file.

Basically, it starts walking through all the files/directories on your system and run md5sum against every file output to a hash output file. If you are working on a Window system, you have md5deep.exe. Same guidelines are true.

md5deep was written by Jesse Kornblum and can be downloaded from: http://md5deep.sf.net

In order to get sorter to work with the hash set you need to index it. To use the hfind tool in TSK, we need to index the database first. This allows for much faster searches. You need to specify the database type. Supported types are md5sum, hashkeeper, nsrl-md5 (index on the MD5 hashes in the NIST NSRL) and nsrl-sha1 (index on the SHA-1 hashes). The NSRL can be indexed with both MD5 and SHA-1.

Future lookups need the database name (it will find the index) and a list of hash values to lookup.

```
# hfind -i <index_type> <hash_database_file>
```

Index Types

```
md5sum
hk (hashkeeper)
nsrl-md5
nsrl-sha1
```

Output: Will create an idx file that corresponds to your database

Examples:

Create a MD5 index file for the known_good_files produced from MD5DEEP output earlier

```
# hfind -i md5sum known_good_files.txt
```

To create an MD5 index file for NIST NSRL:

```
# hfind -i nsrl-md5 /cases/windows/nsrl/NSRLFile.txt
```

So the first thing that we will have to do is run MD5-Deep against a clean version of your operating system. Now this is where difficulty comes into play.

In this case, we actually have already given you and have already created your known good list for you, prior to this incident happening.

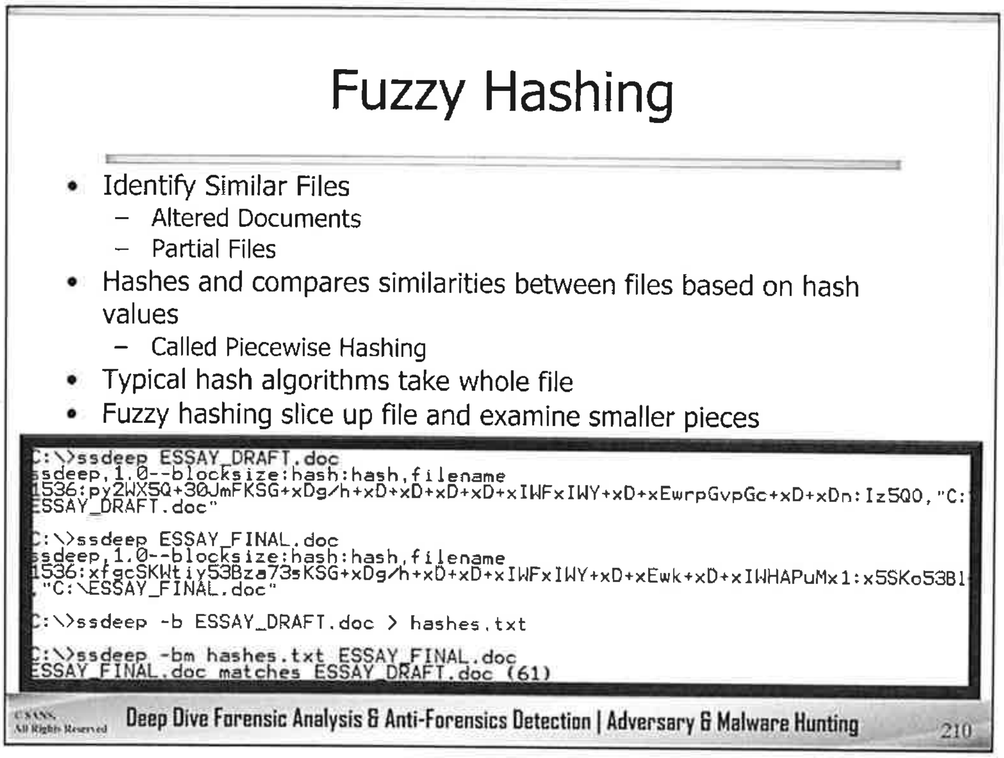
1. Hash all files on a system before an incident:

```
# md5deep -r c:\Windows > E:\winxp-md5.txt
```

2. Create an index file of the known goods file

```
# hfind -i md5sum E:\winxp-hash.txt
```

`Winxp-hash.txt-md5.idx` will be created as a result



One of the inaccuracies an investigator would face is when he encounters a situation where a file is similar to another file. How can you accomplish this? By performing a new hashing algorithm called context triggered piecewise hashing (CTPH). MD5 can only say if a file is exactly matching or if it is different. With CTPH using ssdeep, you can compare two files that are different ,but are close to being the same.

This is the list of the options available to you to accomplish piecewise matching using ssdeep. In most cases, you would use the –m option to perform pattern matching. Without it, you will just acquire a piecewise hash of a file, which is the first thing you need to do to compare it to a similar file.

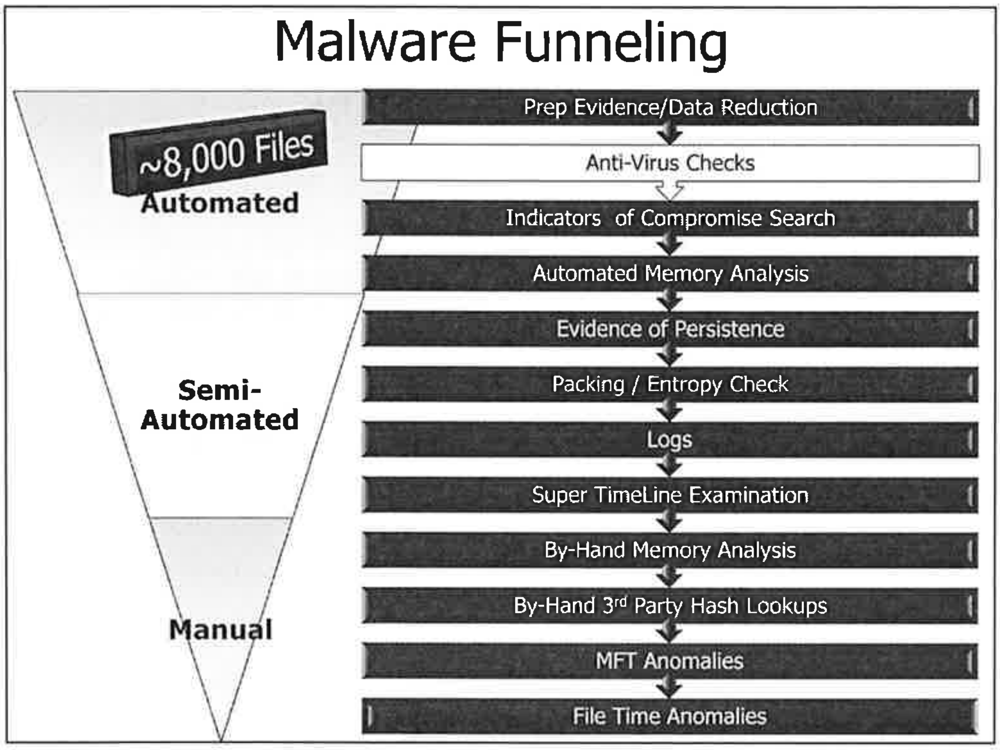
1. Collect piecewise hash of a file or files and write it to a file. # ssdeep file.doc > file_hash.txt
2. Compare against similar file. #ssdeep –m file_hash.txt new_file.doc

Written by Jesse Kornblum

 # ssdeep -m *file_of_hashes [OPTIONS] FILES*

[Useful Options]
-m *file_of_hashes:* Load file of hashes for matching
-r: Enable recursive mode
-p: Pretty matching mode
-d: Enable directory mode
-b: Strips leading directory information
-l: Displays relative file path
-s: Silent Mode: Errors are not listed

Website: http://ssdeep.sourceforge.net/
GUI front-end: http://forensicszone.com/SSDeepFE/SSDeepFE.htm

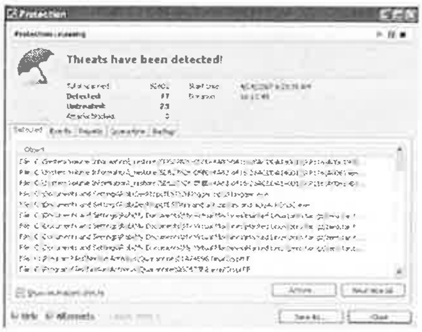


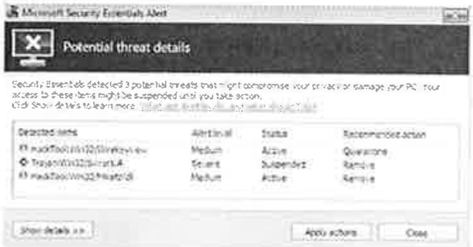
This page intentionally left blank.

Anti Virus Scanner(s):

Deep Dive Tool
Long Processing Time

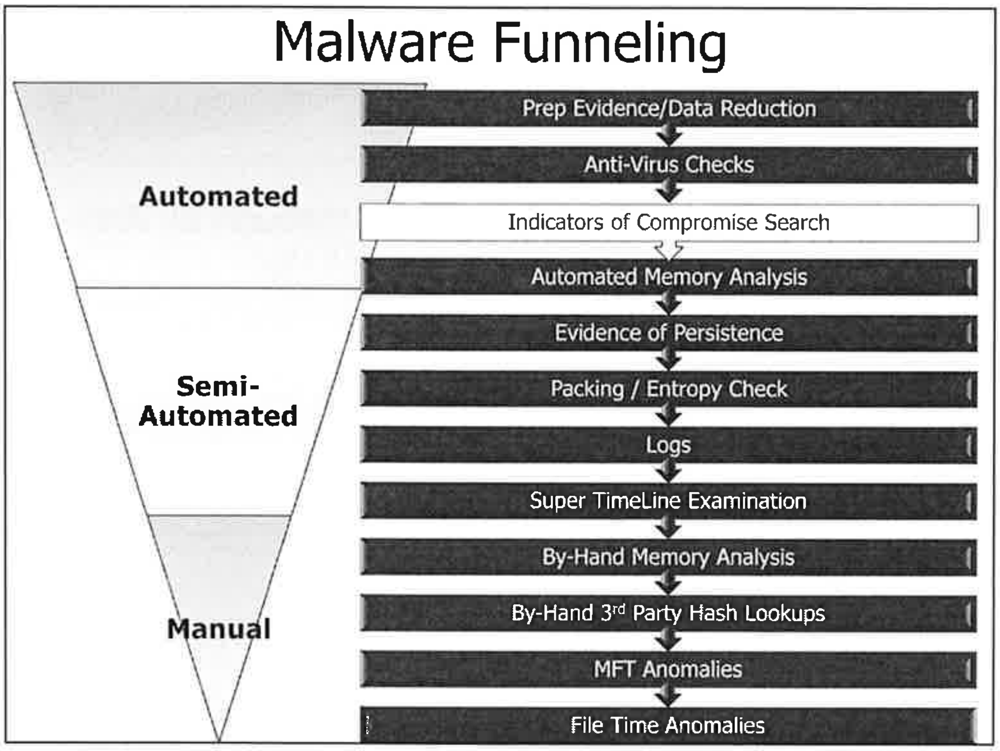
- Run the remote mounted drive through an Anti-Virus Scanner with the latest updates



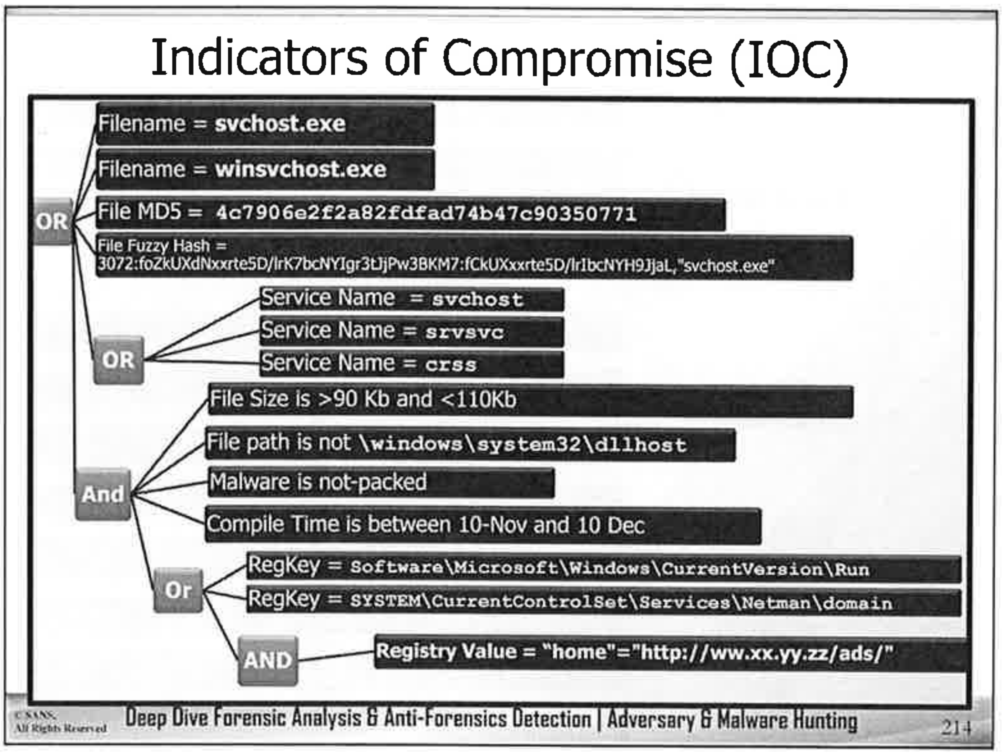


Anti Virus scanners are also another tool you should quickly run on your system when you mount the drive for analysis. Anti-virus scanners are also useful in incident response verification as it can quickly parse through your file system looking for malicious files. However, ensure that your anti-virus software is configured correctly or it may automatically quarantine the evidence you are searching for.

Since A/V is consistently updated, some malware signatures may have been added to it since the incident began. In addition, it is best advised to use several A/V scanners and not rely on one. Set the scanner to accomplish a "deep" scan of the files on the system. Typically I would not only scan the mounted system, I would scan the sorter output directory to ensure you pick up any recovered deleted files during the sorter pass.



This page intentionally left blank.

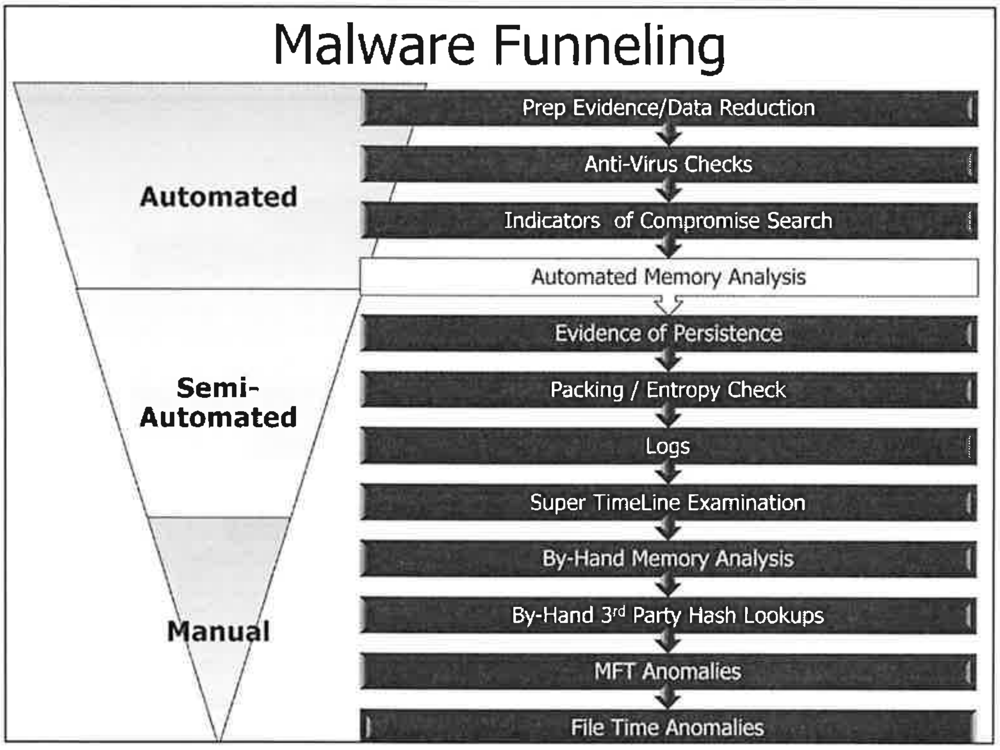


What is an indicator of compromise? It is a very powerful technique to identify malware components on a compromised host. Generally it is a combination of boolean expressions that can be used to identify general characteristics of malware. If these characteristics are found, then you have a hit.

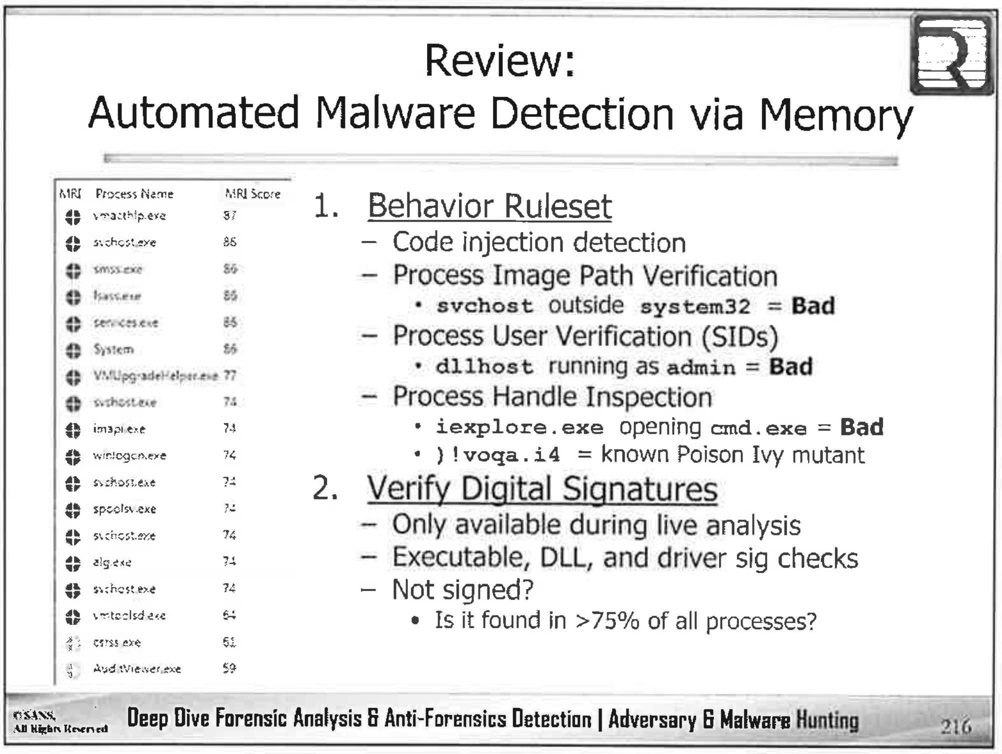
There are two types of indicators: 1. Host based (like you see above). 2. Network based (similar to snort signatures plus additional data).

It is equivalent to narrowing down a suspect through identifying specifics about the suspect. Male. 6'2" ~230 lbs Shaved Head. Blue eyes. Driving a Red or Orange Nissan Xterra.

Your indicators of compromise usually are created by reversing malware and through application footprinting. Some professional groups that are responding to incidents have massive IOC lists that range in the thousands of indicators collected from previous intrusions that they have collected. Malware exhibits many of these signatures and it is up to your team to use these, once detailed, to identify which additional system that might have also been compromised. IOCs are the difference between having to analyze each system in-depth and by analyzing a few in-depth and using that data to identify similar machines on your network that have the same characteristics.



This page intentionally left blank.



The Malware Risk Index (MRI) is a prominent feature in Redline, and a fantastic innovation. The idea behind MRI is that if we have all of these heuristics to identify a bad process, why not create a means to automatically check them instead of relying upon the analyst to remember them all? You can think of this feature as a first pass on the memory image from a junior analyst. It won't catch everything, but it can catch the most obvious anomalies. In the words of co-creator Peter Silberman from the Mandiant blog [1]:

The goal of this feature is twofold. First it is going to help pinpoint specific processes that should be investigated further while attempting to eliminate some of the non-suspicious processes and get them out of the analyst's way. It's also designed to try and make malware detection easier. A lot of work went into looking at samples and how they behave etc., and coming up with definable behaviors that trap those little creatures. MRI is made up of two components. The first component is a definable behavior rule set that is completely customizable. Each process is given a score out of 100. The higher the score, the more likely that it is evil.

How is the score created? The score is created by two components: 1. Behavior rule set. 2. Verification of digital signatures.

The behavior ruleset is made up of three different types of rules:

1. Process Path Verification – allows users to define what processes should be launched from what directories. This triggers on malware that copies and names itself after svchost or other system processes to subdirectories within system folders. For example a default rule is that svchost can only be executed from \windows\system32. Any time we see it running from somewhere else we flag the process.

2. Process User Verification - This triggers on malware spawning svchost for purposes of unmapping image bases or hiding dlls within spawned svchost. So, for example, if malware copies itself to

system32\dllcache and then names ss User Verification – allows users to define what processes should be running under what itself svchost.exe, you can define a rule saying svchost.exe should be running as local service, network service, or system. When Audit Viewer see svchost running as administrator it gets flagged.

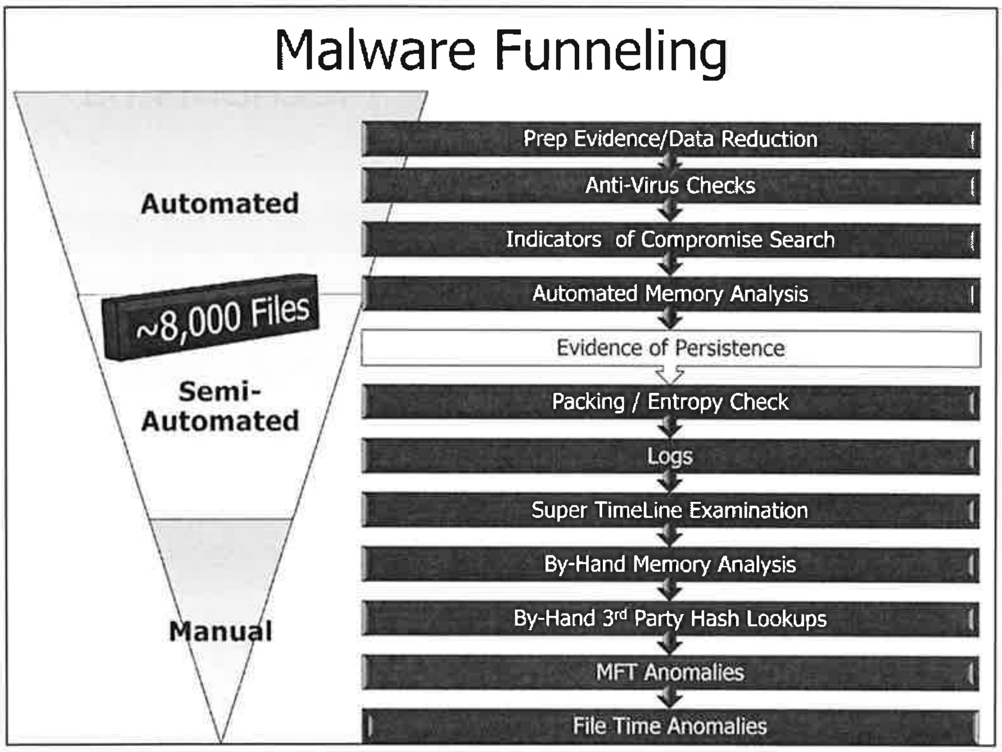
3. Process Handle Inspection – this allows you to define specific rules pertaining to malware or generic behavior. For example a default rule is to flag svchost or iexplore anytime it has a process handle to cmd.exe. There is just no good reason for this to *ever* happen. You can also define rules based on specific malware, for example if [the] "a3c" mutant is present then flag the process as being infected with [the virus] "sality".

The second component of MRI is a process address space scoring mechanism. The new release will contain bug fixes as well as a new feature called "Verify Digital Signatures." When this parameter is turned on, [Redline] will perform a "digital signature check" on all loaded modules. This can only be enabled on live memory analysis. The digital signature check verifies the module on disk is digitally signed. We do a bunch of math and use our Least Frequency of Occurrence to trust modules that aren't signed but occur in more than X% of processes.

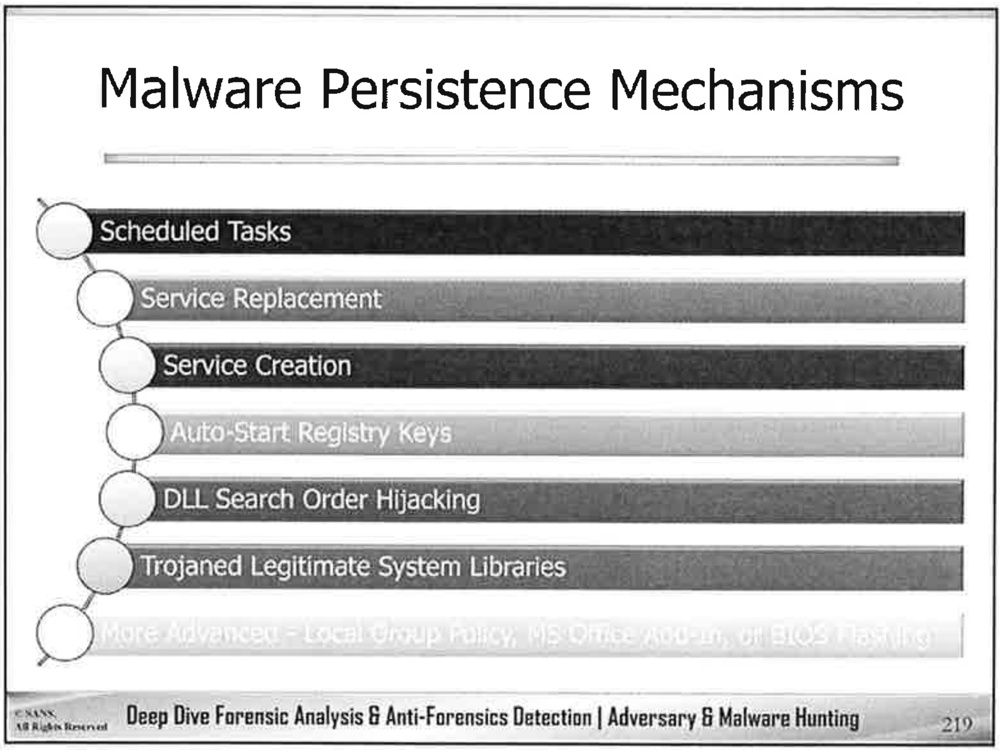
In addition to all of the checks discussed in the Mandiant blog, Redline now also checks for the following:

- Unmapped processes (evidence of code injection)
- Processes started by command shell
- DLL load order / hijacking detection
- Expected command line arguments

[1] https://blog.mandiant.com/archives/741



This page intentionally left blank.



Malware persistence is common using many different techniques that are covered fully in any hacker class. In digital forensics, we only need to know what to look for, not how to implement it. Granted the best hackers end up being the best forensicators in intrusion cases due to their background in how to compromise machines. The ability to be able to ensure your malware survives across multiple reboots would probably cover at least a ½ day of material in a full hacking course.

The most often used capability to achieve persistence is through scheduled tasks using the "at" command. In many cases, an adversary would create a service with their malware or replace an existing service with the new malware. The next most popular malware persistence mechanism is using the registry auto-start at boot or login mechanisms. Using a tool called `autorunsc.exe` discussed on Day 2 will easily parse the autostart locations across scheduled tasks, services, and registry keys. The key to identify malware is looking for non-Microsoft programs that are not signed. Autoruns output here is key to quickly aiding you to identify potential persistence mechanisms still active on the system.

When malware attempts service replacement they will create a service with identical description, ImagePath, service dll, file size, and even mask the strings in the binary as a legitimate exe or dll. Even a little more difficult to find is a new service that sounds like a normal service setting the service name as benign and the ImagePath or ServiceDLL as the malware itself. [1]

There are additional persistence mechanisms and more being discovered being used by our adversaries monthly. Some of the latest techniques include DLL Search Order Hijacking. In this mechanism, a program that is executed will first search in the local directory where the .exe program exists prior to going to the path variables for the .dll file such as system32.[2] If a hacker places a trojanized dll inside that directory, it will be loaded prior to the intended dll found originally in system32. This is incredibly difficult to detect in most cases and there are no known auto-detection methods to look for this. To get the full story on how this technique is used, I would refer you to the Mandiant blog that first publically disclosed this technique. [3]

Some newer techniques also include using local group policy to help run scripts at logon/logoff. These files in the local group policy are only created when the administrator sets them up. However, if a hacker places a script there instead, the group policy will naturally also execute it. This technique is detailed from the HBGary blog found here. [4]

Malware could be installed as a Microsoft Office Add-in. When MS Word Starts, the malware would then be executed. In a situation like that, persistence isn't achieved at boot, but when a program is launched.[1]

Finally, even more advanced techniques include flashing the bios such as the Mebromi malware. This would require advanced skill and is currently mostly unnecessary as our adversaries are able to use much easier mechanisms detailed earlier to hide in plain sight and gain persistence for months prior to discovery. [5]

Windows Services can be configured to recover from failures. Restarting the service is the most common recovery option, but one option that adversaries can use is the "Run a Program" option. [6]

The HEXACORN Blog has a great series about additional locations that can be used for persistent startup. I highly recommend the long series as it is incredibly detailed and shows you exactly how easy it is to achieve persistence through a variety of mechanisms that aren't readily obvious. [7]

References:

[1] http://dfir.sans.org/summit-archives/2010/35-glyer-apt-persistence-mechanisms.pdf

[2] http://blogs.msdn.com/b/larryosterman/archive/2004/07/19/187752.aspx

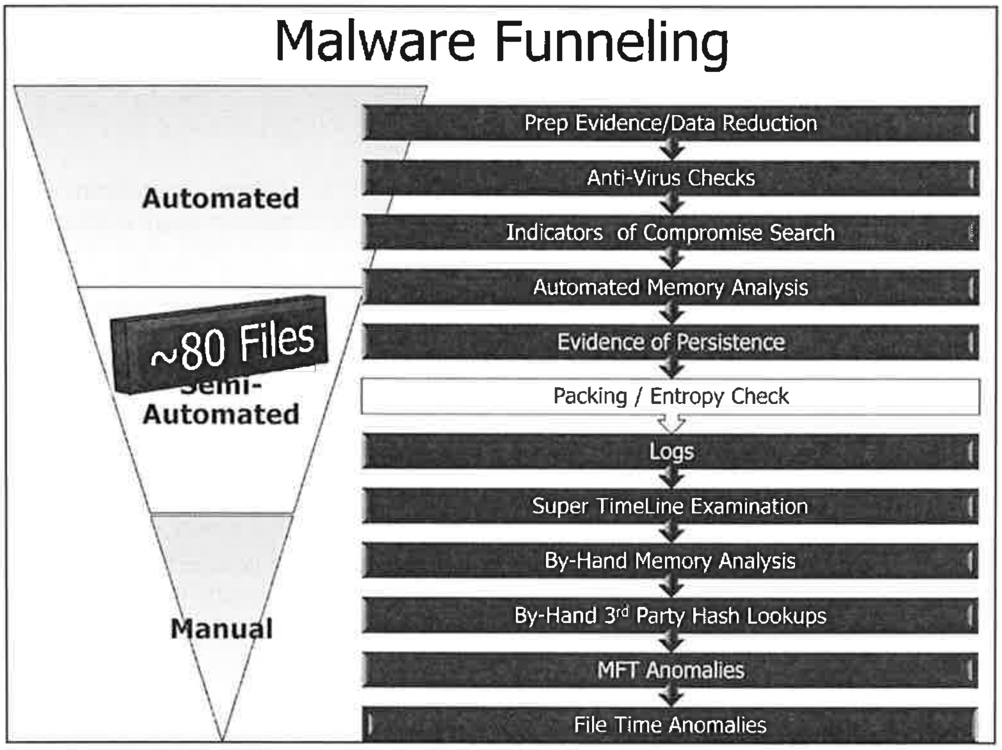
[3] https://blog.mandiant.com/archives/1207

[4] Local Group Policy Malware - http://www.hbgary.com/malware-using-local-group-policy

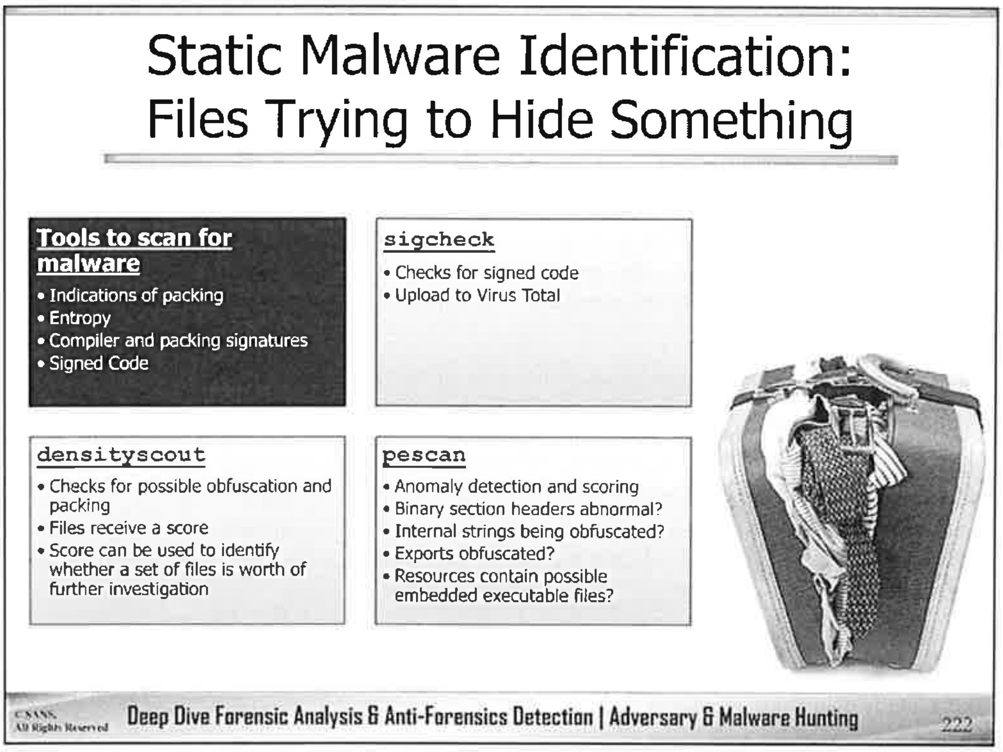
[5] Bios Flashing Malware Mebromi - http://blogs.norman.com/2011/malware-detection-team/mebromi-a-bios-flashing-Trojan

[6] Kansa: Service related collectors and analysishttp://trustedsignal.blogspot.com/2014/05/kansa-service-related-collectors-and.html

[7] http://www.hexacorn.com/blog/2012/07/23/beyond-good-ol-run-key/



This page intentionally left blank.



A common problem facing Incident Responders is identifying suspicious or outright malicious software on a computer system. The Good Guys use a variety of techniques, from Live Response (the examination of a system's state while it's running) to Temporal Analysis (a fancy phrase for "Timelining"), and everything in between. All of these techniques are geared towards identifying how the Bad Guys got in, what they left behind, and whether or not they're still lurking about.

Further complicating this equation is the fact that the Bad Guys don't want to be found, and they usually know a lot about the techniques the Good Guys use to find them. Enter the world of Anti-forensic techniques - a branch of hackery devoted to devising methods to hide things or foil the analytical techniques used by the Good Guys. It's the modern day equivalent of backtracking across your own footsteps or dropping caltrops on the trail while you're being chased. Some of the more common techniques in employ today include:

- Deleting indicators of entry to a system once it's compromised, such as log file entries, file modification/access dates, and system processes.
- Obfuscating running malware by changing its name or execution profile such that it appears to be something benign.
- Storing data on disk in a "packed" format. Packing is a technique that obfuscates or encrypts data or software and encapsulates it in a file along with a program to perform decryption/de-obfuscation. For example, a "Packed Executable" is a piece of software that contains an "unpacking" program and a payload. That payload is often malicious software, such as a virus or Trojan Horse.
- Encrypting data through use of an encryption algorithm and encryption key.

DensityScout is a tool that has been written for one purpose: finding (eventually unknown) malware on a potentially infected system.

Therefore it takes advantage of the typical approach of malware authors to protect their "products" with obfuscation like run-time-packing and -encryption. The tool itself is based on the concept of the Bytehist tool, which is already on-board of REMnux since its initial incarnation.

So what does DensityScout do?

Actually it's quite simple. DensityScout's main focus is to scan a desired file-system-path by calculating the density of each file to finally print out a descending list. Usually Microsoft Windows executables are not packed or encrypted in any way which throws the hits of malicious executables to the top of the list where you can easily focus on.

What's Density?

Christian decided to not use the well-known word "entropy" for the mathematical concept of the calculations going on under the hood of DensityScout. To circumvent any philosophically discussion with real mathematicians it was called "density". But at the end of the day. In his opinion - only one thing counts: Does it work? Yes it does!

pescan is a command line tool to scan portable executable (PE) files to identify how they were constructed. Various metadata is displayed, identifying items such as:

- Compile timestamp
- MACB timestamp
- File size and type of executable
- Target OS and whether binary is 32 or 64 bit
- Linker version used
- Entry point address and desired image base address
- Whether an X509 certificate was used and who the author is
- Whether there is a checksum present and does it match the binary
- Optional analysis of the PE internals to generate an abnormality score which compares the internal construction to the standard operating system files. Higher scores equate to larger differences.
- Optional MD5 and/or SHA1 hashes of the file can be generated as part of the scan.

Computing a hash and Anomaly Detection

Sometimes one wishes to identify a PE file in a way to compare it with other databases that collate binary hashes. To aid in this, ***pescan*** can either compute the *MD5* hash or *SHA1* hash or both using the options ***-md5*** and ***-sha1*** separately or together. The hash is added as a separate field in the output displayed.

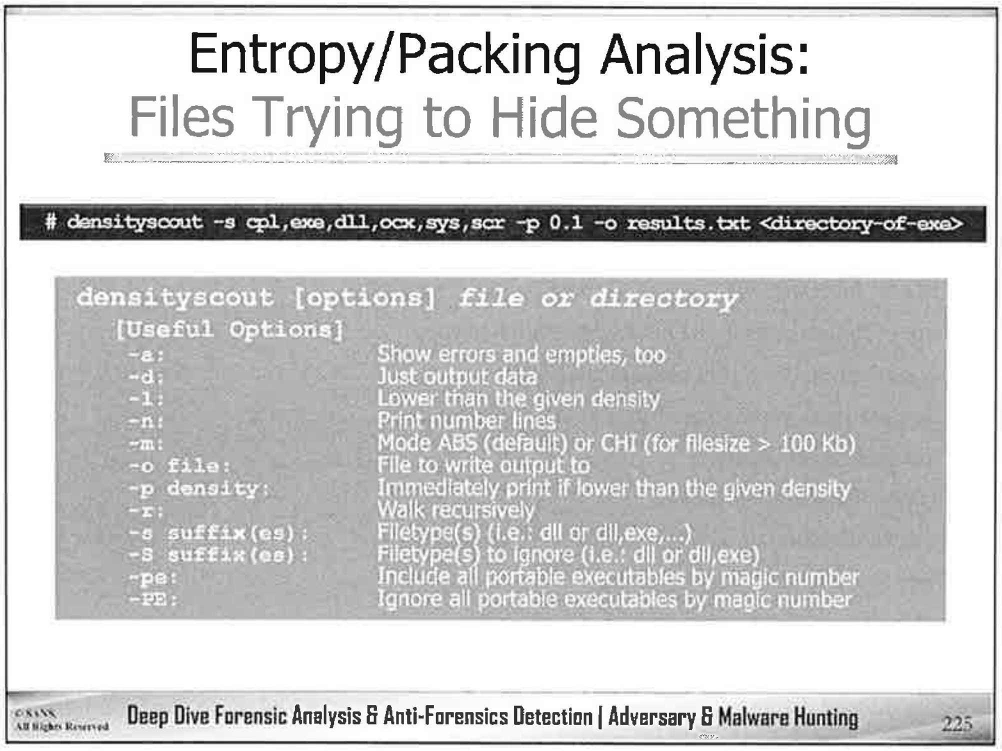
In addition to that hashing options, ***pescan*** has what we call 'abnormality detection'. In the context of ***pescan***, this consists of looking at the PE file structure/packing and comparing it to what normally is available with a Windows operating system executable, dynamic link library or driver file. Based on the differences found, a score is assigned. The higher the score, the more differences were found in the construction of the binary file under analysis from the norm. Furthermore, a high score does not mean the binary is malware or malicious, it just means it was constructed using tools and/or methods that are different than what was normal to other

Windows operating system files. It could be that the author of the binary wanted to obfuscate the internals of the file from reverse engineers, or it could have been built from some non-standard compiler/linker. If there was a high score assigned to a binary, one needs to also look at the reasons for the high score.

Here are a handful of things ***pescan*** looks at during an anomaly scan while making a determination of the score.

- Is the entry point in a standard location or not
- Does the binary make use of thread local storage
- Is the import table present
- Does any of the sections in the binary have high entropy
- Are any of the binary section headers abnormal
- Was there evidence of internal strings being obfuscated
- Were the exports obfuscated
- Was there a checksum mismatch
- Do any resources contain possible embedded executable files
- Was there evidence of binary patching
- Was any of the resources modified after compilation

[1] from tzworks website discussion of pescan -> https://tzworks.net/prototype_page.php?proto_id=15



Here is one of the fastest ways to get a quick glance of if there's anything "suspicious" of a specific Microsoft Windows installation:

```
densityscout -s cpl,exe,dll,ocx,sys,scr -p 0.1 -o results.txt
c:\Windows\System32
```

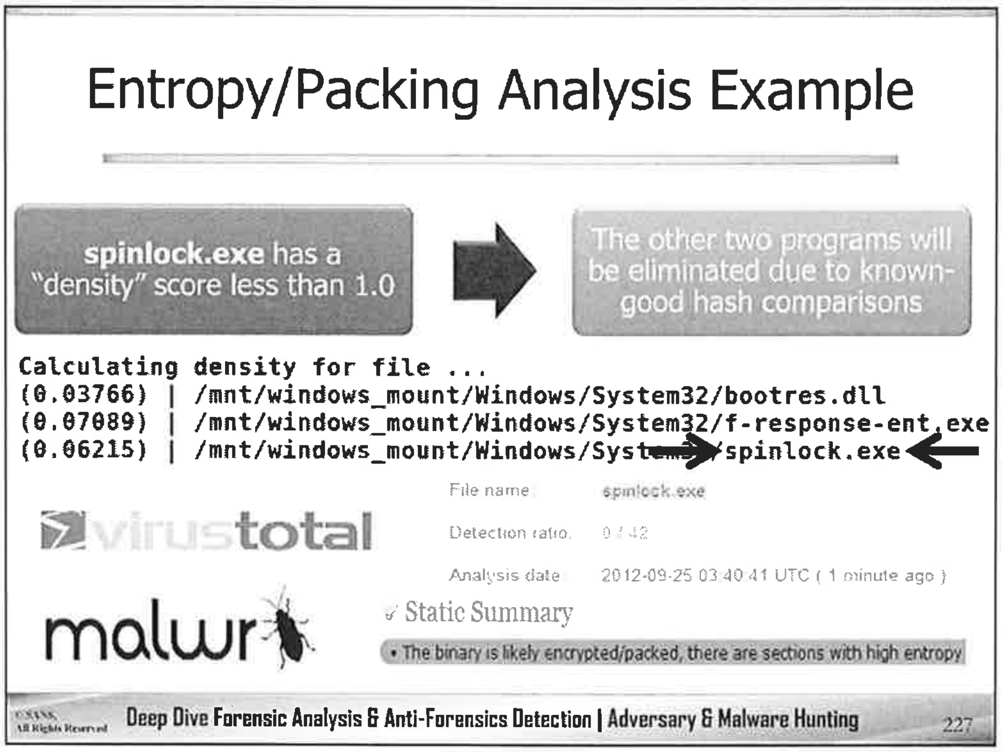
The option "-s cpl,exe,dll,ocx,sys,scr" tells DensityScout to only include files in further computations that have a typical portable executable extension. However, with the latest version (build 42) to achieve this kind of filtering I recommend the all new option "-pe". It tells DensityScout to select the files by checking them against the magic number of portable executables - "MZ", for the ones that do not know. This provides us even with portable executables with extensions we won't expect.

The next option "-p 0.1" is for the impatient ones - like me. With this option you can instruct DensityScout to throw out a hint on the command-line for each file it found with a density below of what you put next to it as soon as it's found. If you do not use this option you have to wait until DensityScout is finished and puts out the desired descending list. Though this option can indeed provide you with the wanted information quite fast the downside on it is obviously that those hints can never be descending. However, the value "0.1" in the shown example is a fairly good threshold to get what you want keeping the hints visually manageable.

The option "-o results" is more or less a no-brain-er. This is the output file which will be used for the result list. If this option is not provided the final list is sent directly to sysout which you might not want because of the quantity. Don't get me wrong, the resulting list is always huge but as already mentioned in the beginning the major advantage is that you only have to focus on the top findings of it.

Last but not least for sure you specify the path to start from. In the current example it's only C:\Windows\System32 without any sub-directories which is focused on. As mentioned this is one of the fastest approaches but a very common one. To do a recursive run you just have to add the "-r" option.

The above is sourced from http://dfir.sans.org/blog/2012/04/26/finding-unknown-malware-with-densityscout



In the above example, we run densityscout against the C:\Windows\System32 directory of one of our compromised systems. Most of the malware we have is not packed, but spinlock.exe was. To show you how spinlock.exe can be detected by looking at the entropy or density of the file we use desnsityscout.

The results are not surprising. The score of less than one shows files that are more interesting than others. We also see several files including f-response-ent.exe that also show a score less than one. However, with a quick md5 hash lookup we can eliminate both bootres.dll and f-response-ent.exe from being suspicious.

Spinlock on the other hand, shows up not as a known file at all, but based on the density, makes it more suspicious. If spinlock was not set as persistent malware at boot, this malware might have been overlooked.

root@siftworkstation:/cases/xp-tdungan-c-drive/sorter/exec# densityscout -r -s exe,dll,ocr,sys,ocx,scr -p 0.1 -o /cases/results.csv ./

DensityScout (Build 42)

by Christian Wojner

Calculating density for file ...

(0.06559) | ./xp-tdungan-c-drive.E01-14460-128-4.exe

(0.07981) | ./xp-tdungan-c-drive.E01-31375-128-4.exe

(0.00460) | ./xp-tdungan-c-drive.E01-10323-128-4.exe

(0.00315) | ./xp-tdungan-c-drive.E01-23296-128-4.exe

(0.09569) | ./xp-tdungan-c-drive.E01-31385-128-4.exe
(0.01405) | ./xp-tdungan-c-drive.E01-23358-128-5.exe
(0.06191) | ./xp-tdungan-c-drive.E01-5237-128-4.exe
(0.01140) | ./xp-tdungan-c-drive.E01-10080-128-7.exe
(0.00484) | ./xp-tdungan-c-drive.E01-8633-128-7.exe
(0.09569) | ./xp-tdungan-c-drive.E01-26050-128-4.exe
(0.07089) | ./xp-tdungan-c-drive.E01-15221-128-1.exe
(0.00387) | ./xp-tdungan-c-drive.E01-584-128-4.exe
(0.03396) | ./xp-tdungan-c-drive.E01-25843-128-1.exe
(0.06215) | ./xp-tdungan-c-drive.E01-7793-128-3.exe ←- SPINLOCK
(0.02004) | ./xp-tdungan-c-drive.E01-9384-128-4.exe
(0.06107) | ./xp-tdungan-c-drive.E01-24041-128-3.exe
(Density) | Filename

spinlock.exe has a "density" score less than 1.0

The other two programs will be eliminated due to known-good hash comparisons

```
Calculating density for file ...
(0.03766) | /mnt/windows_mount/Windows/System32/bootres.dll
(0.07089) | /mnt/windows_mount/Windows/System32/f-response-ent.exe
(0.06215) | /mnt/windows_mount/Windows/System32/spinlock.exe
```

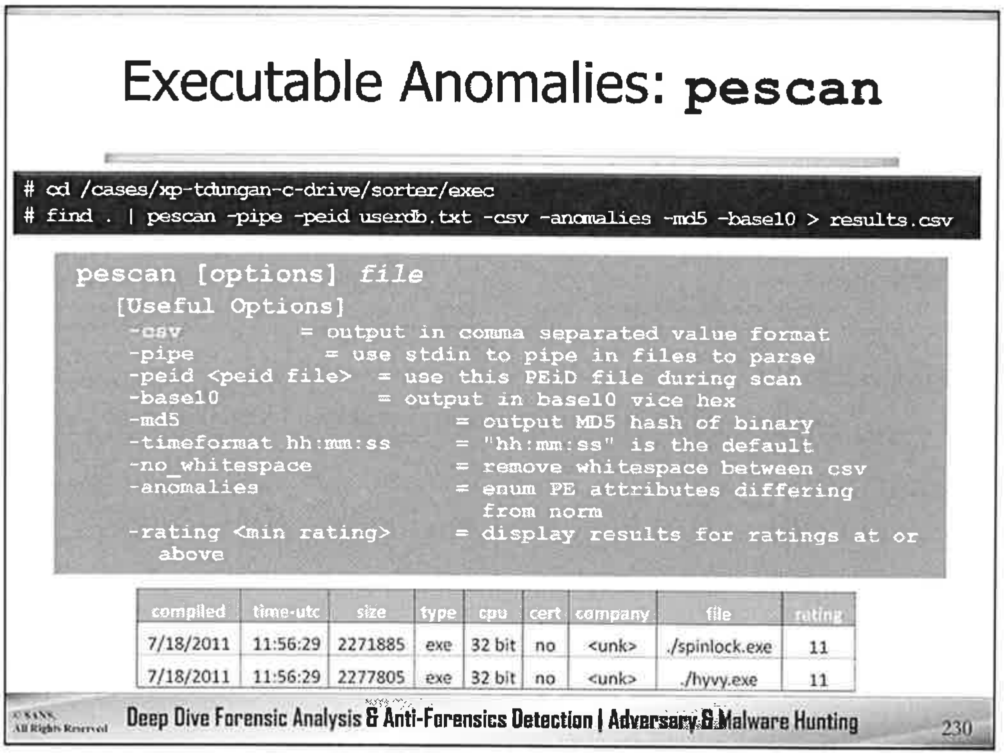
virustotal

File name:	spinlock.exe
Detection ratio:	0 / 42
Analysis date:	2012-09-25 03:40:41 UTC (1 minute ago)

malwr

Static Summary

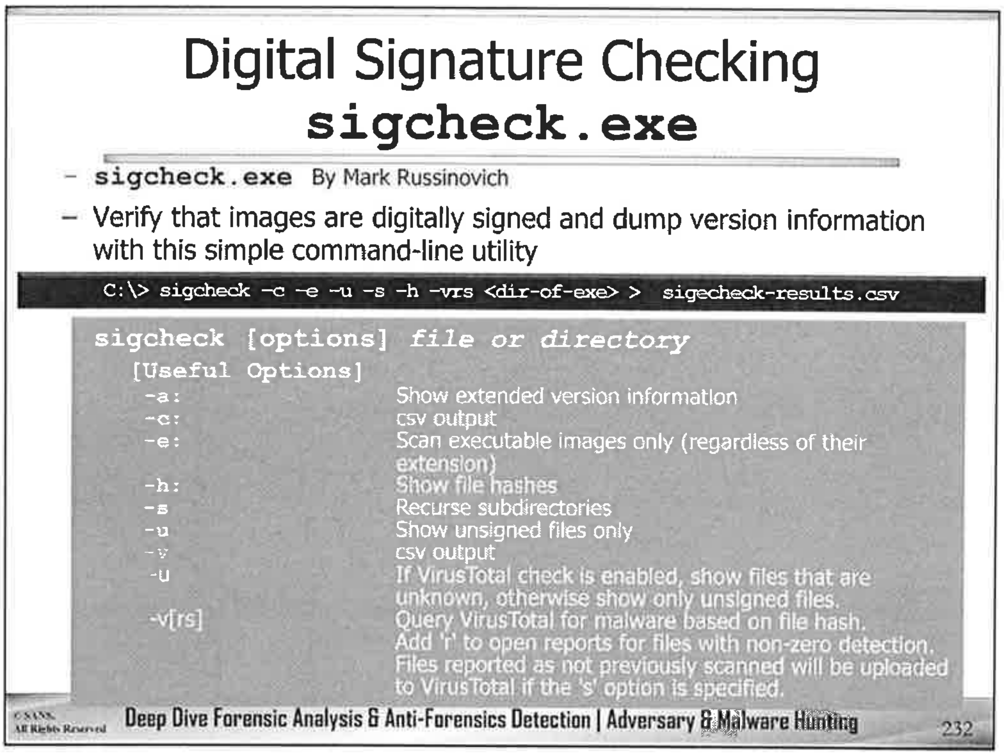
- The binary is likely encrypted/packed, there are sections with high entropy




```
root@siftworkstation:/mnt/windows_mount/WINDOWS/system32# find . | pescan -pipe -peid /cases/xp-tdungan-c-drive/precooked/PEid/userdb.txt -base10 -csv -anomalies -md5 -rating 10

File selected:   /mnt/windows_mount/WINDOWS/system32/spinlock.exe
Company name:    <unk>
Compile date:    07/18/2011 11:56:29 [UTC]
Create date:     04/04/2012 17:06:37 [UTC]
Access date:     04/06/2012 18:58:17 [UTC]
Modify date:     04/04/2012 17:06:37 [UTC]
File size:       0x02271885 [2271885]
PE type:         32 bit - exe
Linker version:  9.0
Min OS version:  Win2K
Entrypoint VA:   0x00008c41 [35905] - FileAddr: 0x00008041 [32833]
Imagebase:       0x00400000 [4194304]
Checksum:        Mismatch

Overall Rating: 11
Note:  Checksum mismatch: 0x00031868 vs 0x00230fbb
Note:  Version information not present
Note:  Debug section not present (or erased)
Note:  Strings may be obfuscated
Note:  At least one section has high entropy
Note:  At least one section has abnormal packing
```

compiled	time-utc	size	type	cpu	min OS	cert	chksum	company	file	rating
7/18/2011	11:56:29	2277805	exe	32 bit	Win2K	no	yes	<unk>	./xp-tdungan-c-drive.E01-**5237**-128-4.exe	11
7/18/2011	11:56:29	2271885	exe	32 bit	Win2K	no	yes	<unk>	./xp-tdungan-c-drive.E01-**7793**-128-3.exe	11
10/24/2011	21:04:14	2000896	dll	32 bit	NT4	no	yes	Apple Inc.	./xp-tdungan-c-drive.E01-27092-128-4.qtx	10
2/22/2012	21:32:30	220672	dll	32 bit	NT4	yes	yes	<unk>	./xp-tdungan-c-drive.E01-13061-128-4.dll	9
2/7/2012	22:58:13	1747456	dll	32 bit	NT4	yes	yes	<unk>	./xp-tdungan-c-drive.E01-31398-128-4.dll	9

```
root@siftworkstation:/cases/xp-tdungan-c-drive/sorter/exec# find . | pescan
-pipe -peid /cases/xp-tdungan-c-drive/precooked/PEid/userdb.txt -base10 -
csv -anomalies -md5 -rating 7 > /cases/possible-malware.csv

# cat /cases/xp-tdungan-c-drive/sorter/exec.txt
</snip>

C:/WINDOWS/system32/hyvy.exe
  PE32 executable (GUI) Intel 80386, for MS Windows
  Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01  Inode: 5237
  MD5: 0159fbbdaff9291425ab976dbbd40aa0
  Saved to: exec/xp-tdungan-c-drive.E01-5237-128-4.exe

C:/WINDOWS/system32/spinlock.exe
  PE32 executable (GUI) Intel 80386, for MS Windows
  Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01  Inode: 7793
  MD5: 6bff2aebb8852fc2658b9768d2166ece
  Saved to: exec/xp-tdungan-c-drive.E01-7793-128-3.exe
```



sigcheck is a favorite tool out of the Sysinternals tool bag. The main and simple purpose of the tool is to check for the code signing of executables in a directory recursively. By using several of the options, namely, -u and v, an analyst can quickly dump out a comma separated value file to a file with only the unsigned entries listed.

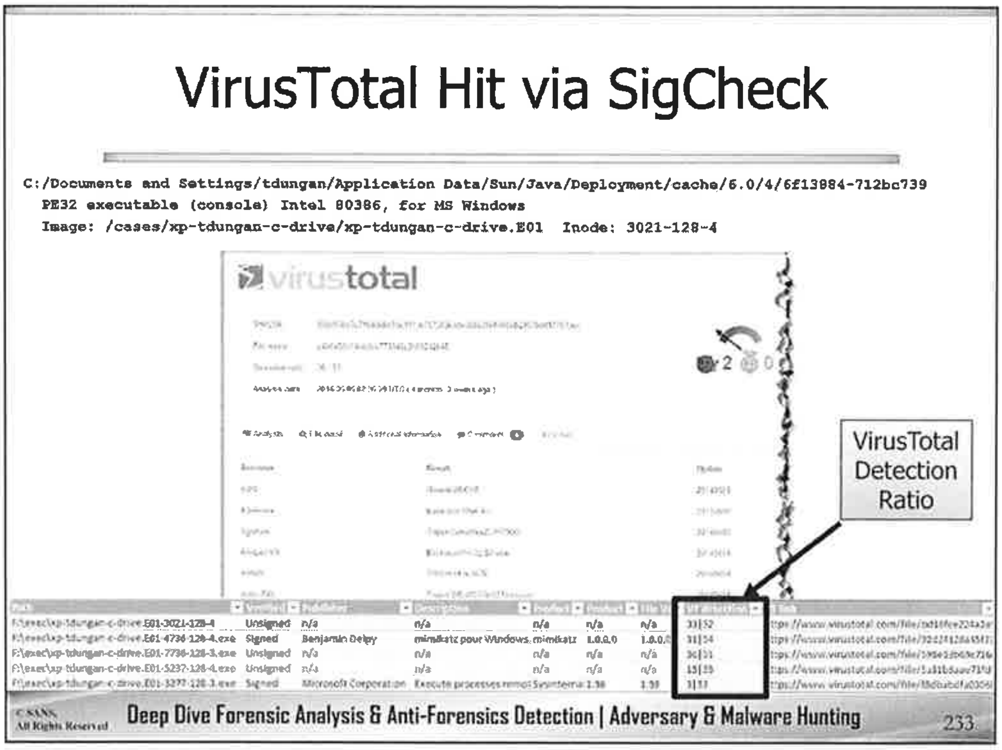
While there are many examples of highly advanced malware with signed code with stolen certificates. Good examples, are flame and stuxnet. Having said that, the majority of malware in the wild are not signed. The reason certain malware is signed is to make it even harder and more difficult to detect, but there is a cost. If you detect the malware and the code certificate is revoked, it could reveal the malware in other locations. So it is a catch-22. If you sign you code, you can make it more difficult to find, however, once found, you expose every system currently infected with it once the cert has been revoked.

The second reason code signing is hard for malware authors? It is fairly difficult to officially sign code using authentic certificates. It is likely that many certificates could be stolen or faked technically to sign code.

As a result, it is likely that much of the malware you will encounter will not be signed. However, the examples of signed malware will continue to increase over time as more advanced malware will be discovered. Using a simple tool like sigcheck can help you easily find the unsigned executables in a directory. If you combine this with densityscout and elimination of known good hashes using sorter, it is likely you will have very few executable malware-ish candidates left in your directory of executables that we started with.

F:\exec>sigcheck -c -e -u -s -h -vr * > sigcheck.csv

Sigcheck v2.1 - File version and signature viewer
Copyright (C) 2004-2014 Mark Russinovich
Sysinternals - www.sysinternals.com



The sigcheck virus total capability is equivalent to running your executables directory through 55 different Anti-Virus engines. The results in static based analysis are fairly stunning although this could take a long time as each file needs to be searched on the VirusTotal website. Plan for this to run overnight for best results. Also recommend that you run this against your filtered, sorter directory after you have removed known_good hashes.

C:/Documents and Settings/tdungan/Application Data/Sun/Java/Deployment/cache/6.0/4/6f13884-712bc739
PE32 executable (console) Intel 80386, for MS Windows
Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01 Inode: 3021-128-4
MD5: 7a0fcd15ee1c2d9d196ab6515adf2f87
Saved to: exec/xp-tdungan-c-drive.E01-3021-128-4

C:/Documents and Settings/vibranium/Local Settings/Temp/a.exe
PE32 executable (GUI) Intel 80386, for MS Windows
Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01 Inode: 7736-128-3
MD5: c4b0458c04abdaa773348c2668212b45
Saved to: exec/xp-tdungan-c-drive.E01-7736-128-3.exe

C:/WINDOWS/system32/hyvy.exe
PE32 executable (GUI) Intel 80386, for MS Windows
Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01 Inode: 5237-128-4
MD5: 0159fbbdaff9291425ab976dbbd40aa0
Saved to: exec/xp-tdungan-c-drive.E01-5237-128-4.exe

C:/WINDOWS/system32/pe.exe

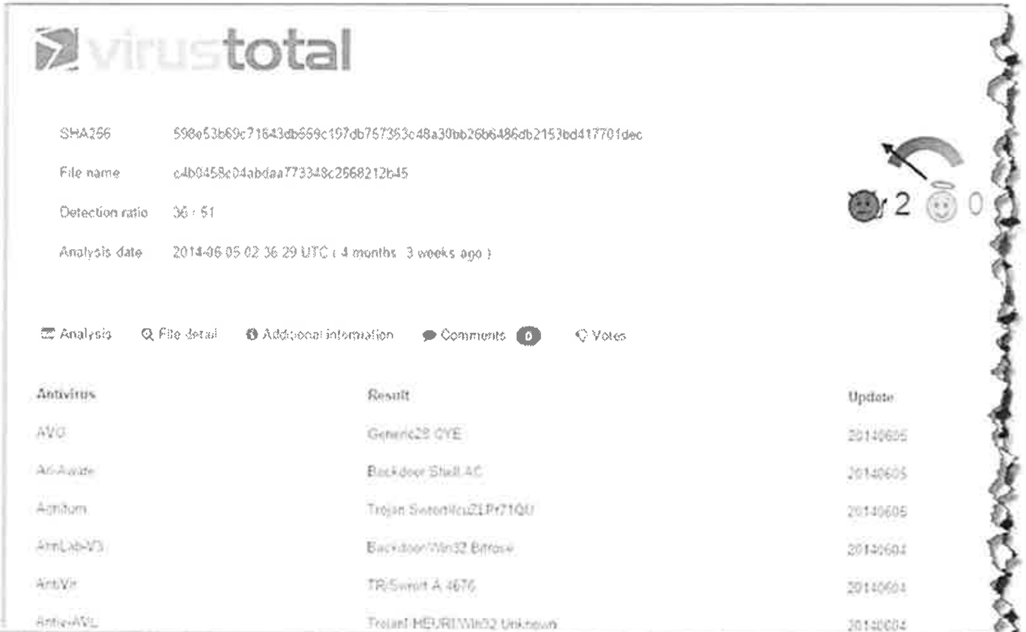
PE32 executable (console) Intel 80386, for MS Windows

Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01 Inode: 3277-128-3

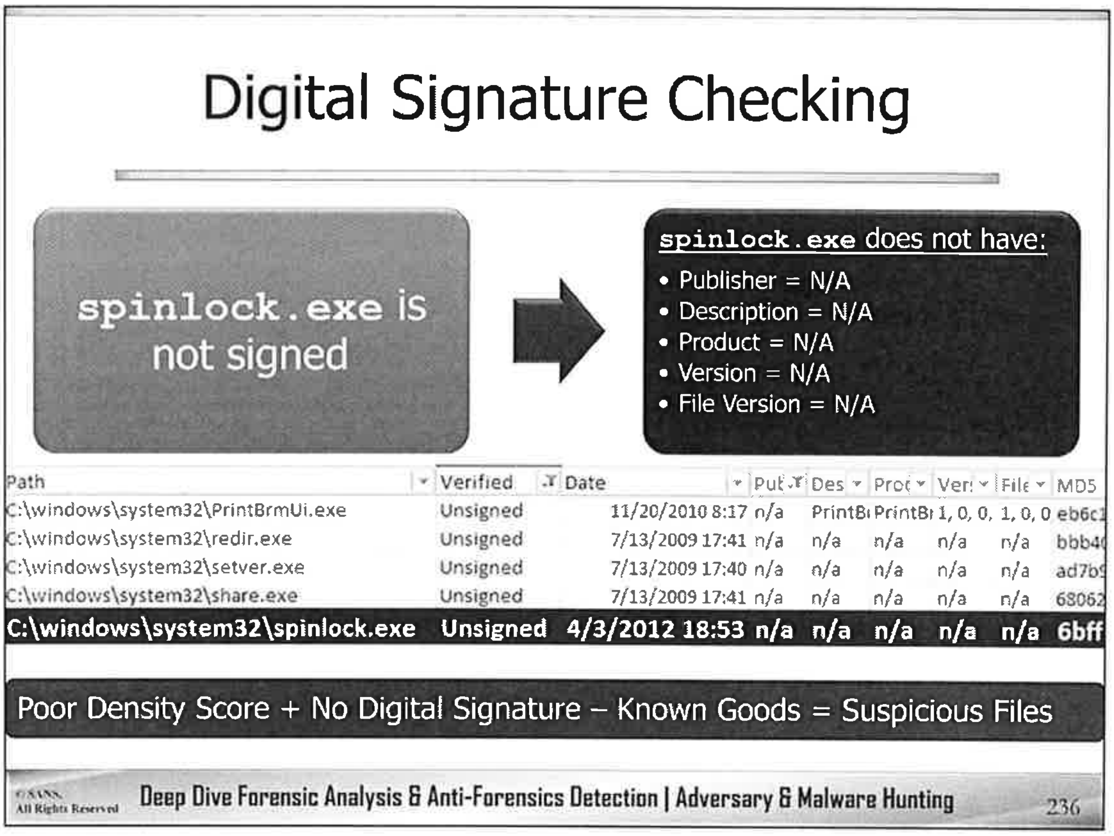
MD5: aeee996fd3484f28e5cd85fe26b6bdcd

Saved to: exec/xp-tdungan-c-drive.E01-3277-128-3.exe

```
C:/Documents and Settings/tdungan/Application Data/Sun/Java/Deployment/cache/6.0/4/6f13884-712bc739
  PE32 executable (console) Intel 80386, for MS Windows
  Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01  Inode: 3021-128-4
```

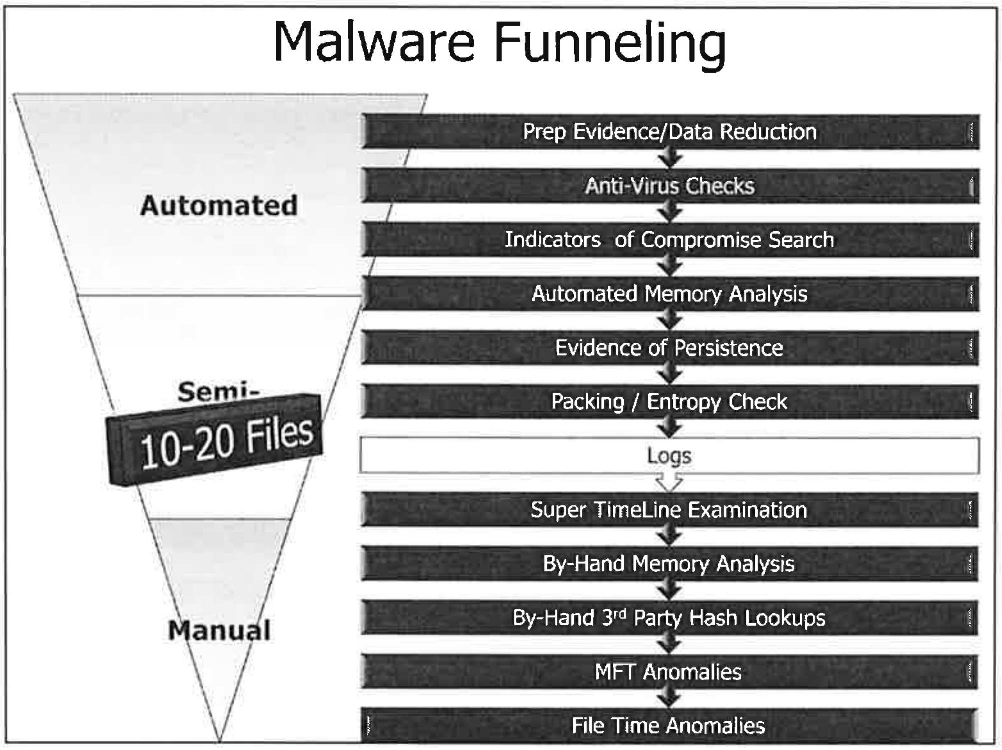


Path	Verified	Publisher	Description	Product	Product	File Ve	VT detection	VT link
F:\exec\xp-tdungan-c-drive.E01-3021-128-4	Unsigned	n/a	n/a	n/a	n/a	n/a	33|52	https://www.virustotal.com/file/bd16fce224a2e
F:\exec\xp-tdungan-c-drive.E01-4736-128-4.exe	Signed	Benjamin Delpy	mimikatz pour Windows	mimikatz	1.0.0.0	1.0.0.0	31|54	https://www.virustotal.com/file/92d24128a45f3
F:\exec\xp-tdungan-c-drive.E01-7736-128-3.exe	Unsigned	n/a	n/a	n/a	n/a	n/a	36|51	https://www.virustotal.com/file/598e53b69c716
F:\exec\xp-tdungan-c-drive.E01-5237-128-4.exe	Unsigned	n/a	n/a	n/a	n/a	n/a	15|55	https://www.virustotal.com/file/5a31b6aae73fd
F:\exec\xp-tdungan-c-drive.E01-3277-128-3.exe	Signed	Microsoft Corporation	Execute processes remot	Sysinternal	1.98	1.98	3|53	https://www.virustotal.com/file/f8dbabdfa0306

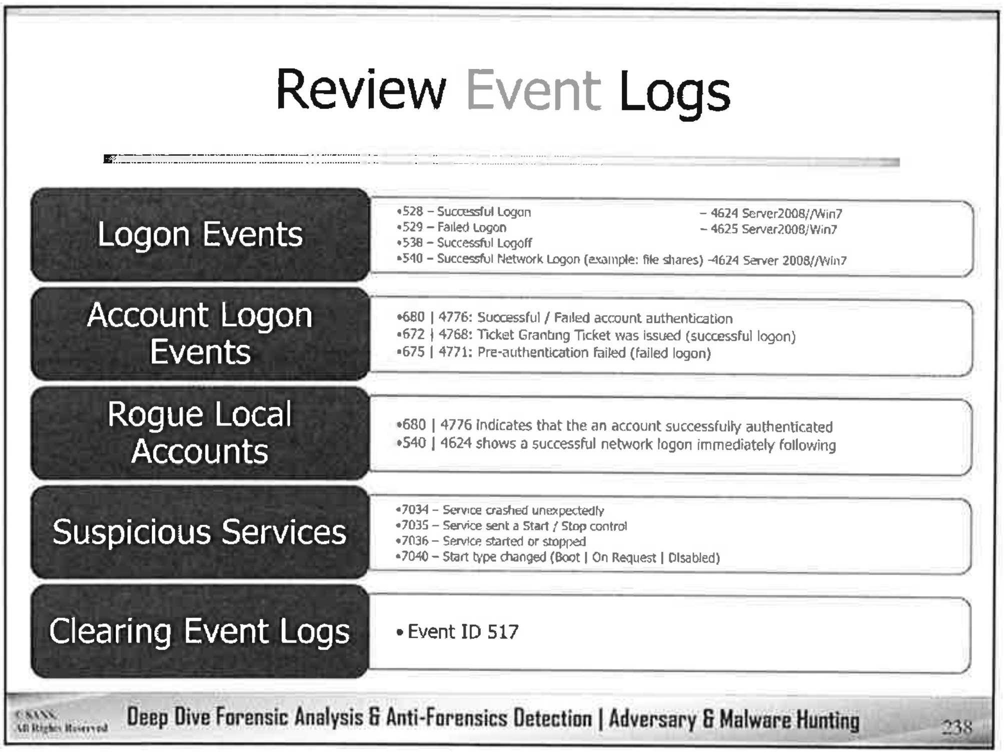


For spinlock, it is already suspicious it had a low entropy/density score. But the file is also not signed. If you subtracted out all the known-good files from the same directory, spinlock would look fairly alone.

Above you can identify another peculiarity with a lot of malware. Malware authors generally do not add publisher, description, product, version, or file version information to their creations. Obvious reasons I'm sure, but they could easily fake this information. Currently, malware usually doesn't come with good coding practices that you usually find in a formal software development shop. While not unique as there is valid software that also doesn't follow this practice, in my experience about 90% of the executables in the system32 directory did. When you cross reference this with eliminating known good files, you are truly on your way to using little hints like this to aid you in seeing possible candidates that could be malware.



This page intentionally left blank.

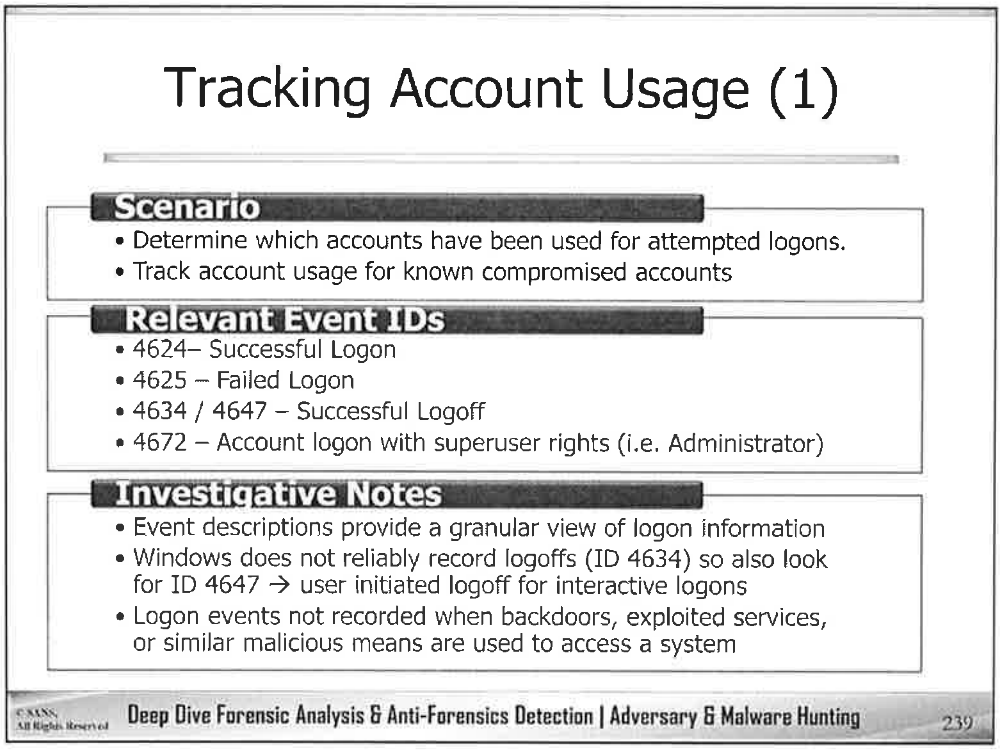


From FOR408 Windows Forensics In-Depth - A big hurdle to understanding event log categories and audit policy is understanding the difference between *Logon Events* and *Account Logon Events*. Most of the confusion stems from the poor name choices. Logon Events refer to login / logoff activity that happens on the actual system being logged into. Thus they are stored locally on that end system. Account Logon Events refer to third-party authorization of credentials provided during that logon session. In a Windows Domain environment, the vast majority of user accounts are actually Domain Accounts, with their credentials stored on the Domain Controller NOT the local system. This is invisible to most users who use the same computer day in and day out. However, behind the scenes, before that user can log onto a workstation in a Domain environment, their username and password must be validated by the Domain Controller using either the NTLM or Kerberos authentication protocol. The Account Logon Events record this process and in this case would be stored on the Domain Controller that verified the credentials. Thus a single user logon can lead to several different events being spread across the workstation (Logon Events 4624, 4625, etc.) and across the Domain Controller (Account Logon Events 4776, or 4768, 4771).

Windows Services control almost every element of the operating system. A service is a process that runs without user interaction and regardless of whether a user is logged in. They are a common target for malware and can provide excellent clues as to malicious behavior on the system you are investigating. Events related to services are located in the System Log, and hence all Event IDs discussed in this section will be for System Log Events.

Services are governed by the Service Control Manager (SCM). It transmits control requests to running services and drivers and maintains status information about those services. The SCM is responsible for updating the System Log with service-related events. When looking for suspicious services, four System Log events are the most useful [2]:

7034: Service crashed unexpectedly – Services should only crash on rare occasions, thus this is an interesting anomaly that may be worth investigating. The event will also indicate how many times the service has crashed previously. Sophisticated attacks like process / dll injection have been known to crash services.

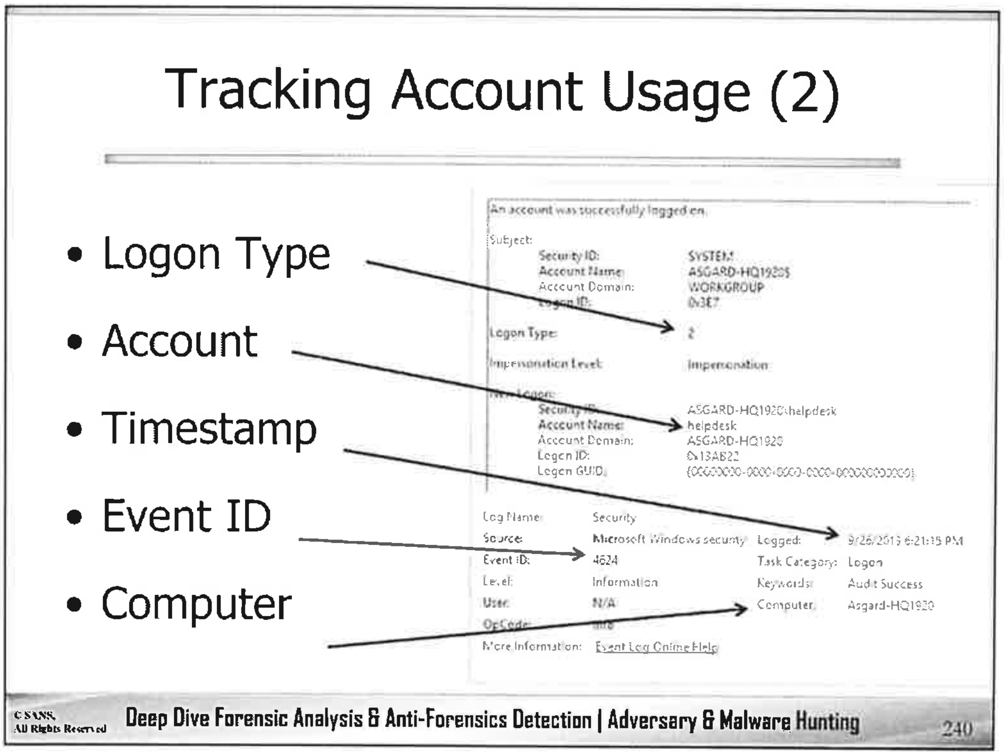


Tracking account usage is one of the more common uses for reviewing event logs. Knowing when a user account logged on to a system and subsequently logged off can provide helpful corroborating evidence along with other forensic artifacts found. If account credentials are suspected to be compromised, reviewing successful logons throughout your network can help track where the hacker has been. We will also see that remote logons are recorded and can provide excellent profiling information on how an authorized or unauthorized user is traversing the network and attempting to authenticate with resources.

With the introduction of Win2008, Account usage Event ID Codes were collapsed into only a handful of possibilities (in comparison XP had 24 Event IDs for logon actions). The most common are the 4624 / 4634 pair, which shows a successful logon / logoff, as well as the time period of the complete user session. Windows is not always consistent with recording logoff events (type 4634), so it is wise to also look for 4647 events (user initiated logoff for interactive sessions). 4625 events indicate logon failures and are often reviewed for evidence of password guessing attacks. Event ID 4672 is recorded for administrator-equivalent logons in addition to the standard 4624 event. The IDs covered on this slide are triggered by a mix of Success and Failure audits. They can be used for justifying the inclusion of both Success and Failure into your Logon Events Audit Policy.

When a hacker gains access to a system through some sort of an exploit (remote code execution, privilege escalation, service exploitation, client-side attacks resulting in backdoors, etc.) there is typically no record of "logon" within the event logs. This is intuitive because a backchannel is being used and the standard APIs for access are being circumvented.

Corresponding event IDs in Windows XP/2003: 528-552



We will be reviewing logs in this section using the built-in Event Viewer tool. It can be opened via the Run command bar or command line using its file name, `eventvwr.exe`. Alternatively, it can be accessed via the Computer Management Microsoft Management Console (MMC) -- <right-click> on My Computer in the Start Menu and select "Manage".

Each record shown in the Event Viewer can be expanded for additional information. When auditing account usage events, we focus on five fields of the event record. Our preliminary information comes from the footer. There we find the timestamp information, providing the date and time that the account logon occurred. The Computer reference tells us the hostname of the system that the event was recorded on. This can be very helpful when reviewing logs from multiple systems simultaneously to find logon patterns. Finally, the Event ID tells us what type of Logon/Logoff event this is, and in this case it is Event ID 4624, which is reserved for successful logons. An excellent reference for a wide range of Event IDs is the following website:

http://www.ultimatewindowssecurity.com/securitylog/encyclopedia/

We can gather further details within the Event Description. Most notably, the Account Name field tells us which account was successfully logged into. In this case it was an account named "helpdesk". The Logon Type field is easy to ignore, but plays a very important role in tracking account usage. There are multiple different ways that a logon can occur in a modern Windows system. It isn't enough to just tell us that the Administrator account successfully logged into this system at 6:21:15 PM on 9/26/2013. We would also like to know if they were using the console, logging in via some network protocol like SMB, or using Remote Desktop. The Logon Type can give us this further information. In this case, Logon Type 2 indicates that the logon was accomplished via a console.

It is important to note that we usually don't just rely upon one event. After gathering information from this 4624 event, we would then review other events surrounding it in addition to looking for a matching 4634 event, indicating that the user logged off from this session.

While these are the most important elements, each recorded event contains additional information defining the event and providing valuable information regarding why that event was triggered. [1] In this slide we are showing these elements using the detailed Event Properties function of the Event Viewer. However, it is important to note that the elements are all stored in a XML format and can be parsed and displayed using a variety of tools. The information available is as follows:

Logged: The timestamp of when the event was triggered. Displayed in **Local System Time**

Level: Provides the reason the event triggered and in some cases its severity

User: The account that triggered the event

Computer: The name of the computer where the event occurred (useful when reviewing logs from different systems simultaneously)

Source: The application, service, or Microsoft component that logged the event

Task Category: The category of the event; in Security logs this will be the Security Event Category managed via the audit policy

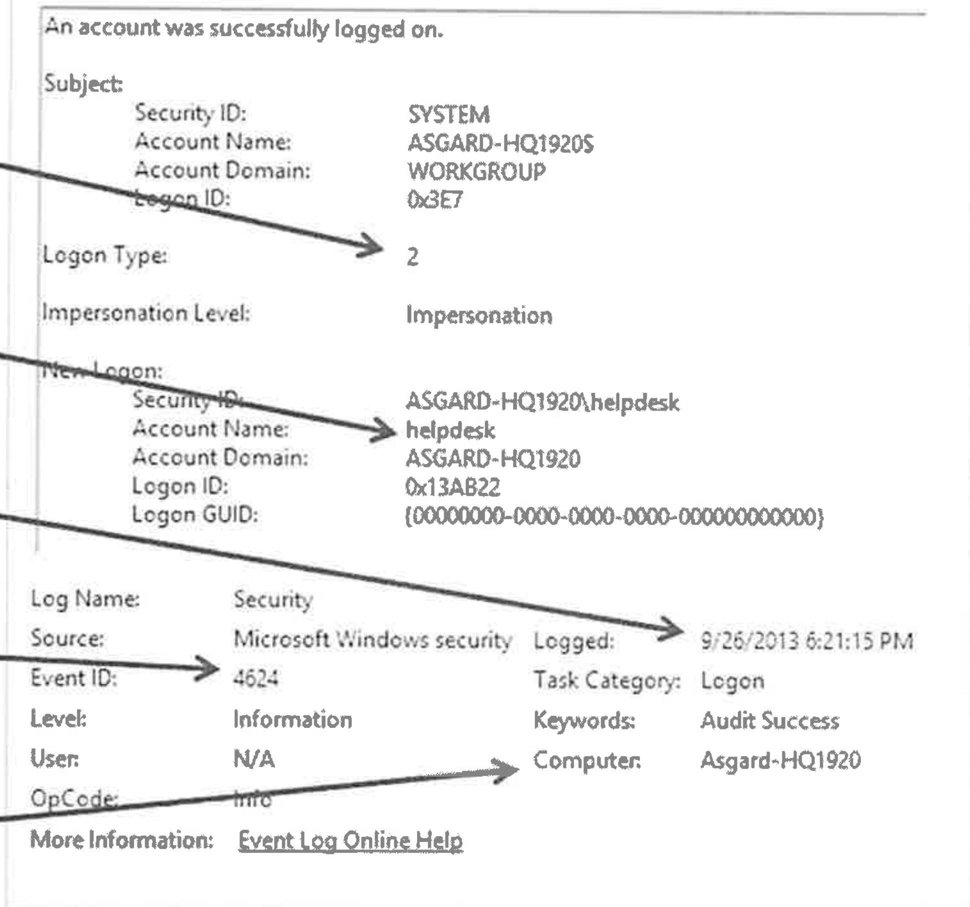
Event ID: A unique code assigned to various system functions. Can often be the most useful indicator of what triggered the event (if you have a good Event ID reference)

General Description: A canned text description of the event, sometimes with additional information. Descriptions can excel at "saying nothing", but can sometimes provide valuable information including remote IP addresses, hostnames, etc.

Details : An optional field that can contain additional raw data (or error codes) generated by the event

[1] http://technet.microsoft.com/en-us/library/bb726966.aspx

- Logon Type
- Account
- Timestamp
- Event ID
- Computer



Logon Type Codes

Logon Type Code	Explanation
2	Logon via console (i.e. using the keyboard)
3	Network logon
4	Batch Logon – Often used by Scheduled Tasks
5	Windows Service Logon
7	Credentials used to lock or unlock screen
8	Network logon sending credentials in cleartext
9	Different credentials used than logged on user – runas command
10	Remote interactive logon (Remote Desktop Protocol)
11	Cached credentials used to logon – system likely offline from DC
12	Cached Remote Interactive (similar to Type 10)
13	Cached unlock (similar to Type 7)

Logon Events can give us very specific information regarding the nature of account authorizations on a system if we know where to look and how to decipher the data that we find. In addition to telling us the date, time, username, hostname, and success/failure status of a logon, we can also determine by exactly what means a logon was attempted. Logon Type Events are provided within the "Description" information of Logon Events and can give us this further information about how a logon was attempted. This information can be exceedingly helpful to an investigation, allowing us to determine whether actual "hands on keyboard" were used to logon or if the session was created using remote means via a Server Messaging Block (SMB) protocol connection or by using something like Remote Desktop. Further, we can use this information to differentiate between system-based logons such as those performed by Scheduled Tasks, and user-based interactive logons.

The following Logon Type Codes can be used [1]:

2: Logon via a console (i.e. using the keyboard)

3: Network logon (often using something like SMB for drive mapping)

4: Batch logon (Scheduled Tasks) – non-interactive

5: Windows Service Logon – non-interactive

7: Lock or unlock of screen

8: Network logon sending credentials in cleartext (potentially indicative of a downgrade attack or older admin tool)

9: Different credentials used to authenticate than those currently logged on with (Runas command or similar)

10: Remote interactive logon (Terminal Services / Remote Desktop Protocol)

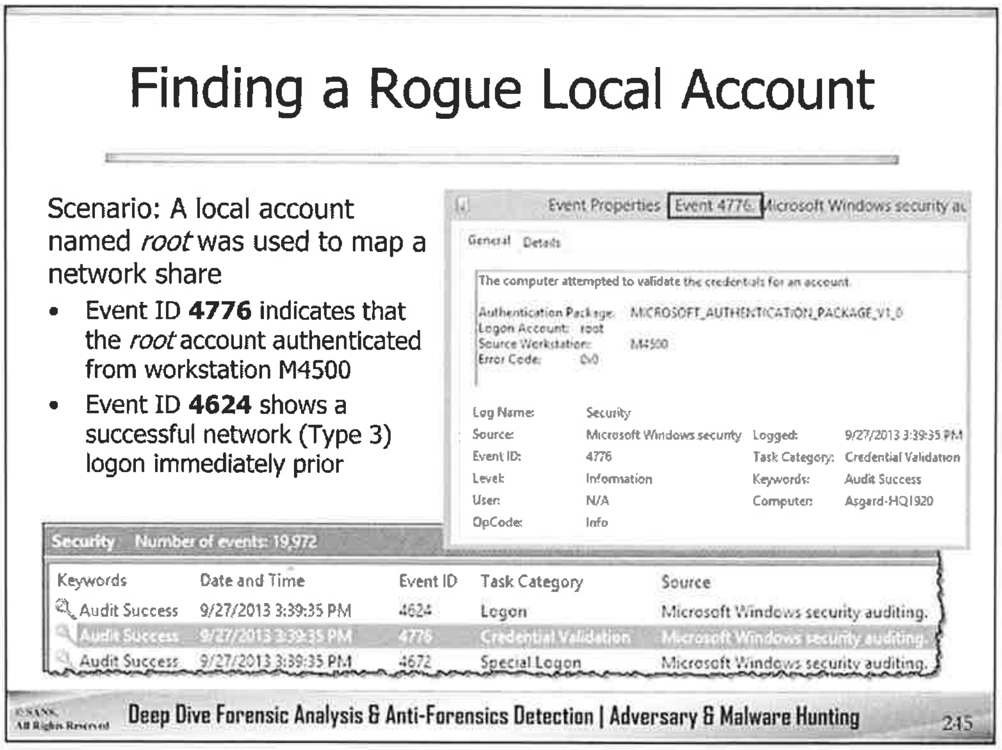
11: Cached credentials used to logon due to system not in communication with the Domain Controller

12: Cached credentials used for a remote interactive logon (RDP). Previously rare, but now being seen when Microsoft "live" accounts are used for authentication on standalone workstations

13: Cached credentials used for an unlock operation

In addition to helping our investigations, knowing these Logon Types can also be an excellent way to audit security within your enterprise. For instance, if you see Logon Type 11 in your event logs, this indicates that cached credentials are being used within your environment. That may be perfectly acceptable for a laptop system, but on something like a server, this could be a dangerous security misstep that could allow an attacker to very quickly harvest additional accounts.

[1] http://www.windowsecurity.com/articles/Logon-Types.html

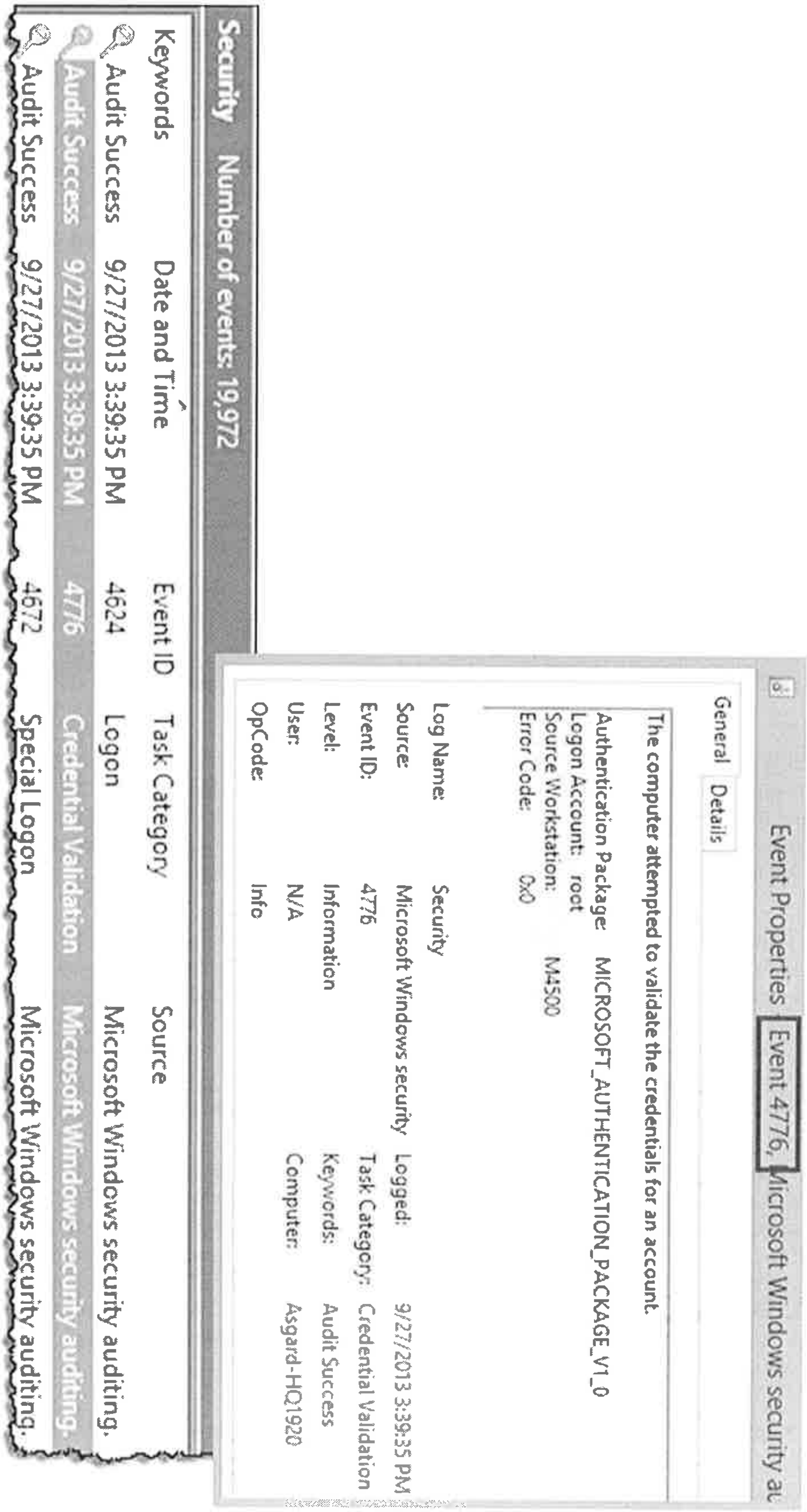


Taking what we know about Account Logon Events, this scenario demonstrates how easy it is to find a potential rogue account via the Event Logs. This scenario assumes that local accounts (particularly privileged ones) are not routinely used to logon within the environment, which is a reasonable assumption in many enterprises.

Doing a quick filter of the event logs for Event ID 4776 (Account Logon Success/Failure) identifies an entry on 9/27/2013 at 3:39:35 PM on computer Asgard-HQ1920. Opening the event shows it was a Audit Success Event and that the logon account used was named *root*. Since the 4776 Event was not located on a Domain Controller, this indicates that *root* is a local account present on the system Asgard-HQ1920. Looking further at the 4776 Event Properties, we see that the originating workstation was named M4500. Now we have information indicating a successful remote access of Asgard-HQ1920 using a local account. Strange....

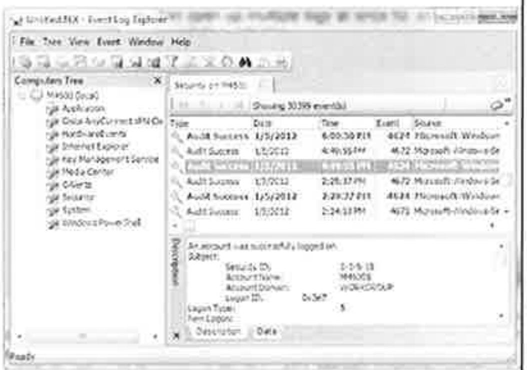
The next step is to review the events surrounding this anomalous 4776 event for further information. An Event ID 4624 was recorded immediately before the 4776 event. While not shown on this slide, a review of the 4624 event showed a Type 3 (network) logon for the *root* account from system M4500. This is indicative of authentication commonly seen during activities like share mapping. Thus it appears we have a remote system (M4500) mapping a share to our workstation (Asgard-HQ1920) using a local administrator account.

Corresponding event IDs in Windows XP/2003: 528,680


Security Number of events: 19,972
Keywords
Date and Time
Event ID
Task Category
Source
Audit Success
9/27/2013 3:39:35 PM
4624
Logon
Microsoft Windows security auditing.
Audit Success
9/27/2013 3:39:35 PM
4776
Credential Validation
Microsoft Windows security auditing.
Audit Success
9/27/2013 3:39:35 PM
4672
Special Logon
Microsoft Windows security auditing.
Event Properties - Event 4776, Microsoft Windows security au
General
Details
The computer attempted to validate the credentials for an account.
Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
Logon Account: root
Source Workstation: M4500
Error Code: 0x0
Log Name: Security
Source: Microsoft Windows security
Logged: 9/27/2013 3:39:35 PM
Event ID: 4776
Task Category: Credential Validation
Level: Information
Keywords: Audit Success
User: N/A
Computer: Asgard-HQ1920
OpCode: Info

Event Log Explorer

- Supports .evt and .evtx formats
- Can open up multiple logs at once for simultaneous searching and correlation activities
  - Merge logs together to correlate
  - Including remote Event Logs!
- Very tolerant of log corruption
  - Standard and direct mode
- Excellent filtering
  - Including by Event Description
  - Right-click for Quick Filters
- Color coding by Event IDs
- Free for personal use



 Deep Dive Forensic Analysis & Anti-Forensics Detection | Adversary & Malware Hunting 247

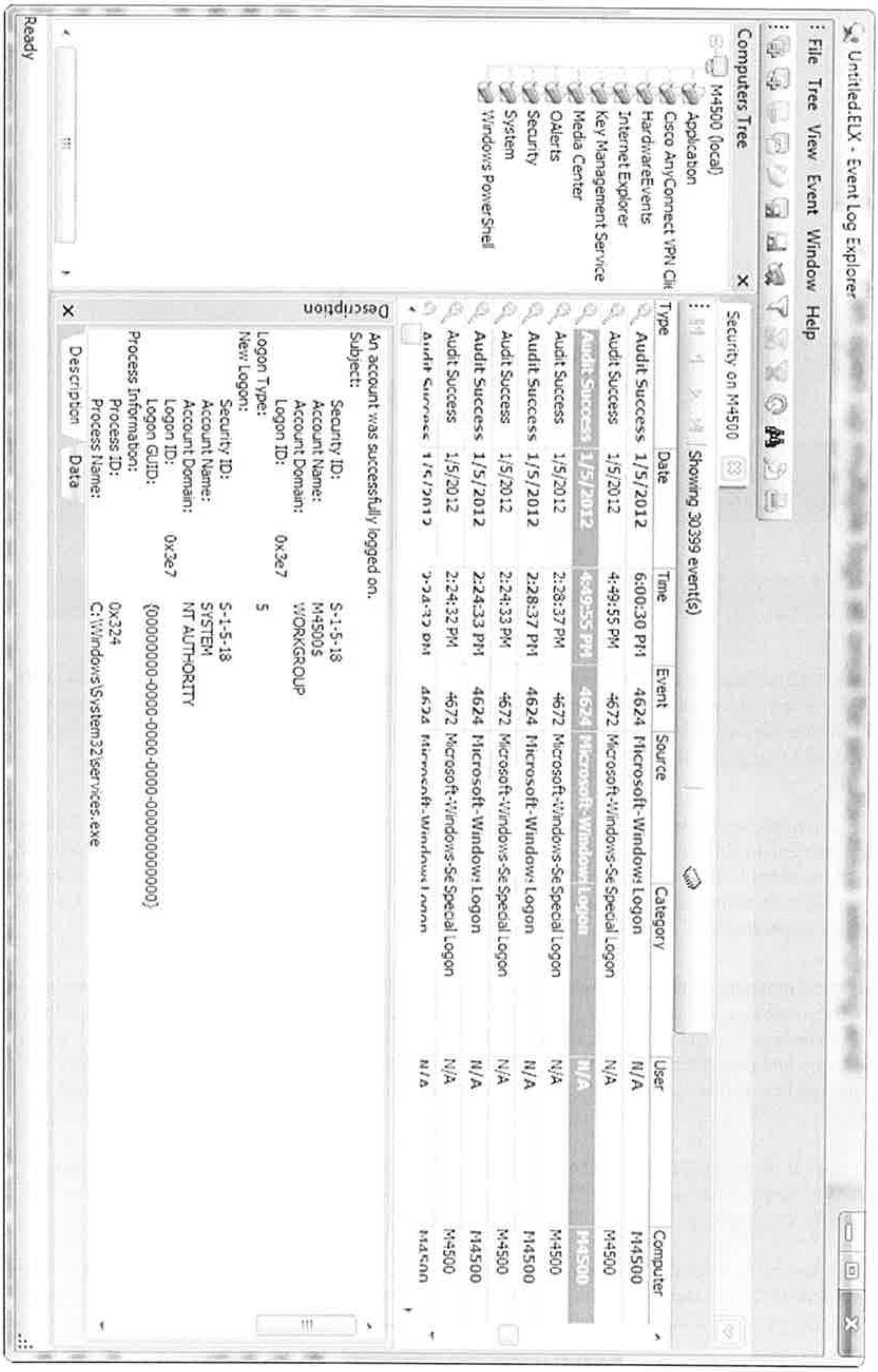
The built-in Windows Event Viewer has some significant drawbacks. It can be very cumbersome to review a large number of events within the interface. There is no capability to load multiple security logs (i.e. from different systems) and filter or search in parallel. It is very intolerant of corrupted logs. Due to the nature of forensic data collection, we often encounter corrupted event logs, and I have seen less experienced forensic analysts give up when those logs could not be opened in Event Viewer! Never fear, we have alternatives.

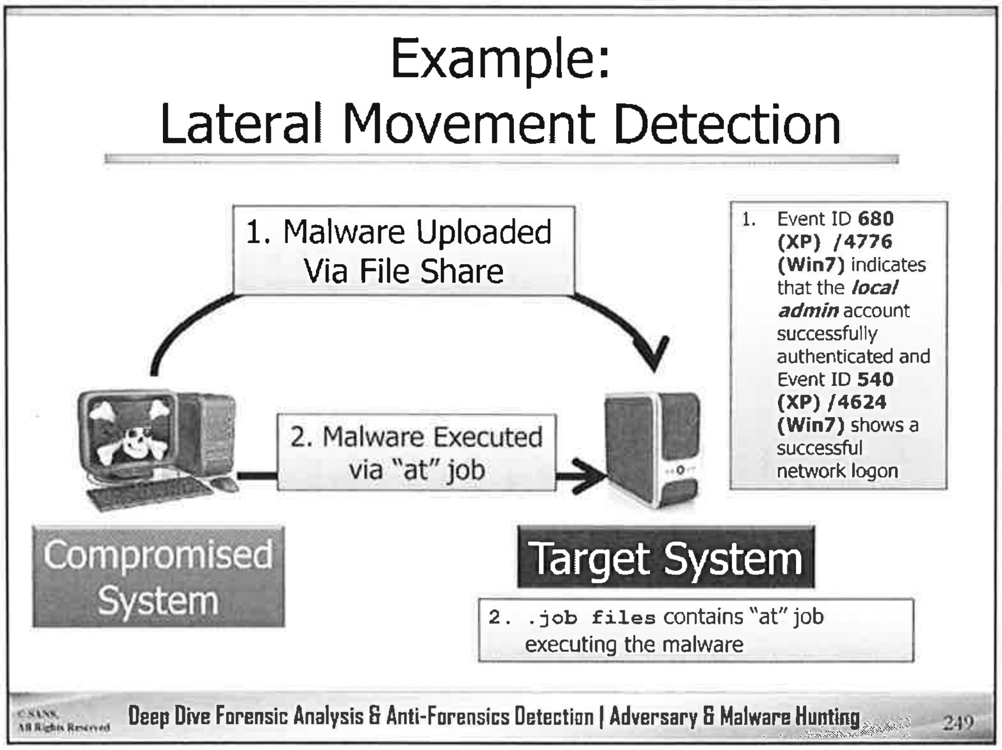
Event Log Explorer is a third-party event log management software package that runs circles around the built-in Microsoft tools. [1] It provides just about every feature that a forensic analyst could want when doing a log review and greatly speeds up the process. It supports logs from every current Windows NT operating system (Windows NT to Windows 2008) and as such can read both .evt and .evtx log formats. One of its biggest benefits is that it is very capable of working with corrupted log files. You can open up a log file in one of two ways: Standard Mode attempts to open up the log using API-like methods and Direct Mode parses the log in raw form. Thus we get the benefits of an offline parser tool combined with a GUI interface and review platform.

Another area where Event Log Explorer shines is in its features that assist log review. It allows many log files to be opened simultaneously and even merged, greatly aiding with event correlation and reducing the amount of time needed to search. This ability extends to being able to open logs on multiple ***remote*** systems simultaneously for live reviews. It has a very robust filtering capability, including access to the text-based Event Description field where so many of our forensic artifacts are located (such as Logon Type in ID 4624 Security Events). Quick Filters (accessed by right clicking on an event column) allow options like showing only events of a specific type or removing types of events from view. Another nice feature is the ability to color code different Event IDs so you can quickly zoom into the places within the log where you want to focus your analysis.

Event Log Explorer is free for personal, non-commercial use, and is approximately $99 for a full license.

[1] http://www.eventlogxp.com/


Untitled.ELX - Event Log Explorer
File Tree View Event Window Help
Computers Tree
M4500 (local)
Application
Cisco AnyConnect VPN Clie
HardwareEvents
Internet Explorer
Key Management Service
Media Center
OAlerts
Security
System
Windows PowerShell
Security on M4500
Showing 30399 event(s)
Type Date Time Event Source Category User Computer
Audit Success 1/5/2012 6:00:30 PM 4624 Microsoft-Windows Logon N/A M4500
Audit Success 1/5/2012 4:49:55 PM 4672 Microsoft-Windows-Se Special Logon N/A M4500
Audit Success 1/5/2012 4:49:55 PM 4624 Microsoft-Windows Logon N/A M4500
Audit Success 1/5/2012 2:28:37 PM 4672 Microsoft-Windows-Se Special Logon N/A M4500
Audit Success 1/5/2012 2:28:37 PM 4624 Microsoft-Windows Logon N/A M4500
Audit Success 1/5/2012 2:24:33 PM 4672 Microsoft-Windows-Se Special Logon N/A M4500
Audit Success 1/5/2012 2:24:33 PM 4624 Microsoft-Windows Logon N/A M4500
Audit Success 1/5/2012 2:24:32 PM 4672 Microsoft-Windows-Se Special Logon N/A M4500
Description
An account was successfully logged on.
Subject:
Security ID: S-1-5-18
Account Name: M4500$
Account Domain: WORKGROUP
Logon ID: 0x3e7
Logon Type: 5
New Logon:
Security ID: S-1-5-18
Account Name: SYSTEM
Account Domain: NT AUTHORITY
Logon ID: 0x3e7
Logon GUID: {00000000-0000-0000-0000-000000000000}
Process Information:
Process ID: 0x324
Process Name: C:\Windows\System32\services.exe
Description Data
Ready



One of the easiest techniques that adversaries use to move laterally from one compromised system to another targeted system is to use windows file shares to upload the file to the remote machine. In many cases, the attackers will use the local admin account on a machine in order to move through a domain as quietly as possible to avoid logging on the domain controller.

Once the session has been established, generally using a simple "net use" command, uploading the malware to the remote machine in a directory of their choice is accomplished. Once the malware has been successfully copied to the remote machine an "at" command is sent from the compromised system to the target system to execute the malware immediately or at a later time. It can also be used to establish the malware as a service on the remote target machine.

Tracking lateral movement in an enterprise is one of the more difficult tasks an investigator may take on. Without good event logs, the task is near impossible. In this example we see an adversary pivoting from a compromised internal system to a target machine. File shares are a common attacker methodology used to upload malware and access data as quietly as possible. If the attacker has discovered a local administrator account that works on other systems, they can be even more stealthy by avoiding the domain controller altogether.

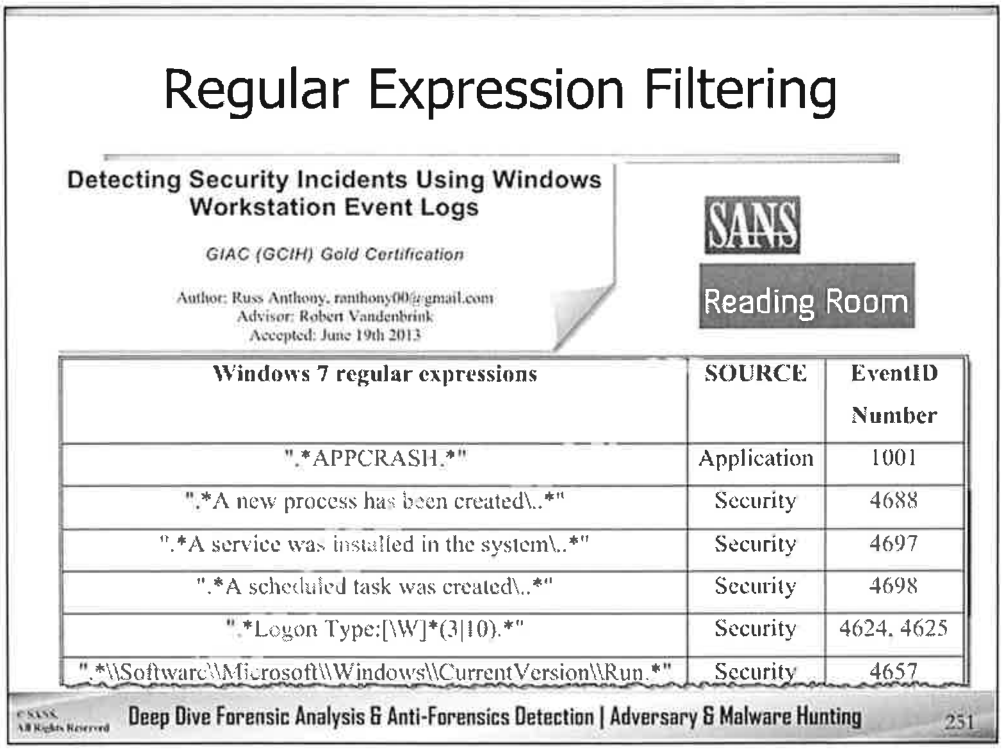
The good news is that regardless of the lateral movement technique used, logs can be used to piece events together. At some point the attacker may have authenticated with the compromised system so we can find evidence on the compromised system (in the case of a backdoor, logon events could be absent).

If a local account is used to authenticate with the target system, logs will only be present on the target. This should be a wakeup call to security administrators who do not aggregate logs from workstations and instead solely rely upon review of domain controller logs! In our example here we would see a Account Logon event on the target (Event ID 4776) showing successful authentication along with a Logon event showing the type of account logon (mounting a file share is a network logon and hence would record Event ID 4624). Looking for this pattern on workstations is a powerful technique for identifying strange activity.

Finally, if the attackers used a domain account to mount that remote share, then the domain controller will record a ticket-granting ticket (Event ID 4768) authenticating the account and a service ticket (Event ID 4769) identifying and providing access to the remote target machine (assuming Kerberos authentication). On the target side, a single network Logon event is recorded (Event ID 4624).

This is a great example of the "embarrassment of riches" that we sometimes have when good auditing policies are in place. Very commonly a simple action by a user can cause multiple event log entries to by recorded throughout the network. When reviewing Event Logs in a networked environment it is very important to understand all the locations where logs may reside. To get the full picture of activities (and when they occurred), it may be necessary to recover logs from multiple systems.

Corresponding event IDs in Windows XP/2003: 528, 540, 672, 673, 680



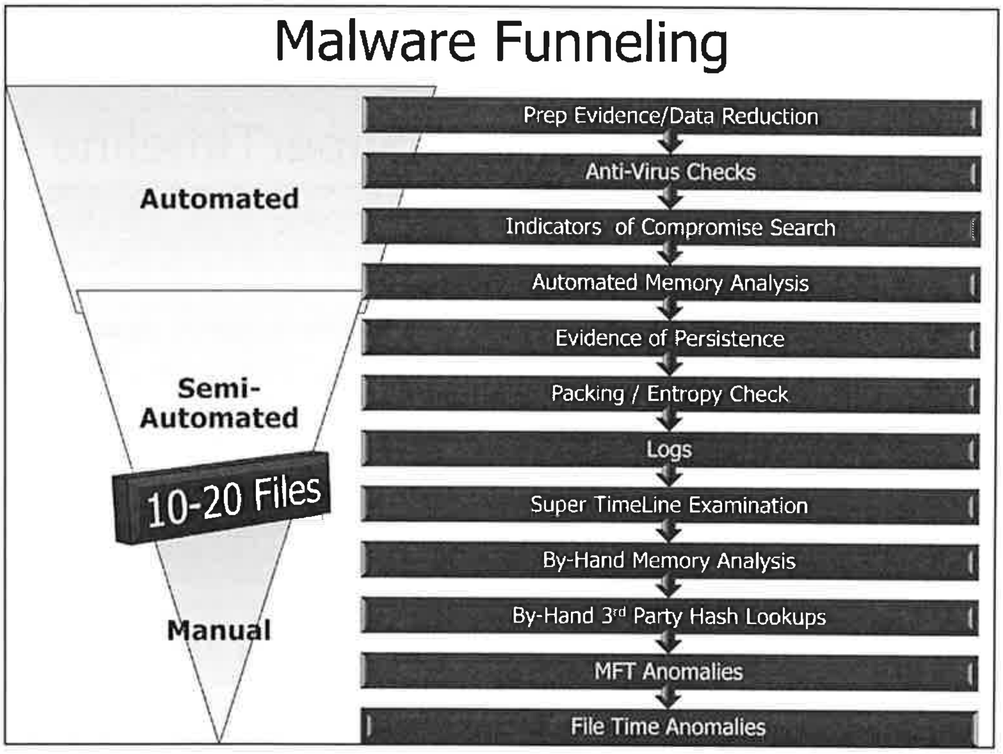
Windows 7 regular expressions	SOURCE	EventID Number
".*APPCRASH.*"	Application	1001
".*A new process has been created\..*"	Security	4688
".*A service was installed in the system\..*"	Security	4697
".*A scheduled task was created\..*"	Security	4698
".*Logon Type:[\W]*(3|10).*"	Security	4624, 4625
".*\\Software\\Microsoft\\Windows\\CurrentVersion\\Run.*"	Security	4657

A great resource you should be aware of is the SANS Reading Room. Here you will find a wealth of papers written by analysts, graduate students, and attendees working towards their GIAC Gold Certification (as we see on this slide). Every time I venture into the Reading Room, I find something relevant to my day-to-day job. A fantastic recent paper was written by Russ Anthony, who delves deep into just how valuable Event Logs are for detecting system anomalies and investigating security incidents. [1] Of particular note is a recommended collection of regular expressions for filtering logs. In this section we have taken a very targeted approach, largely looking at very specific types of events and only a handful of Event ID types. A complementary analysis technique is to search events of every type for targeted keywords for whatever actions we are investigating. While not an exhaustive list, enough examples are provided to aid in the creation of your own regular expression filters. Once you have your filters ready, tools like Event Log Explorer will allow you to search for those regular expressions. Alternatively you could build X-Path filters to integrate directly into the Windows Event Viewer.

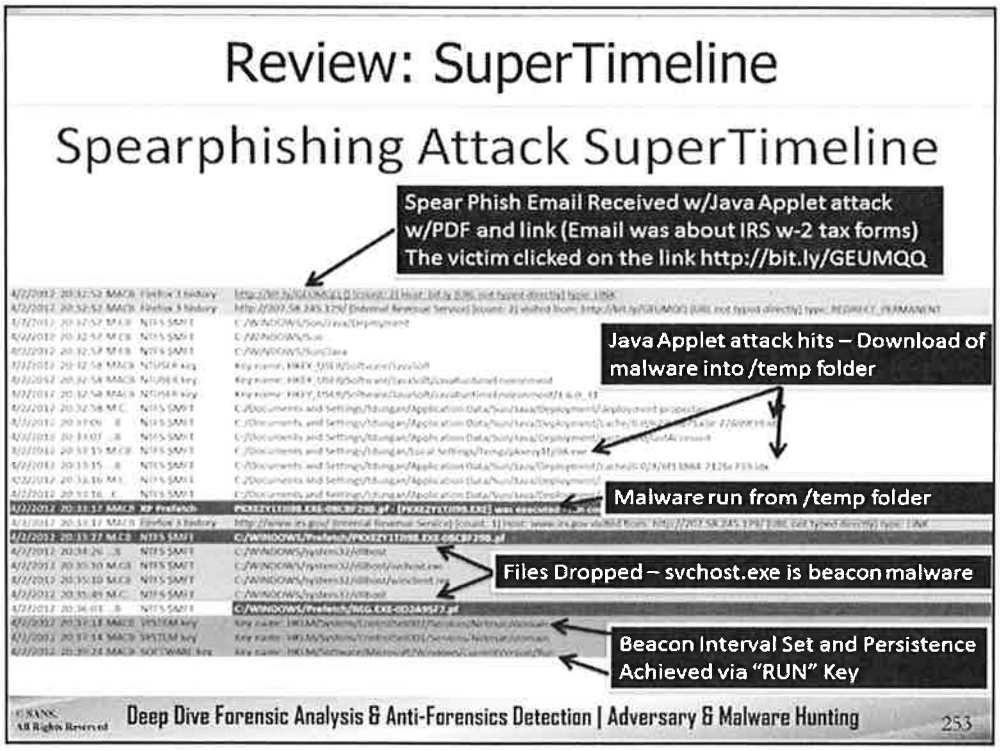
See /Documents on your USB for more log info

- Web Server Logs – IIS (USB /Documents)
- Windows Firewall Logs (USB /Documents)
- Using LogParser (USB /Documents)

[1] http://www.sans.org/reading-room/whitepapers/logging/detecting-security-incidents-windows-workstation-event-logs-34262

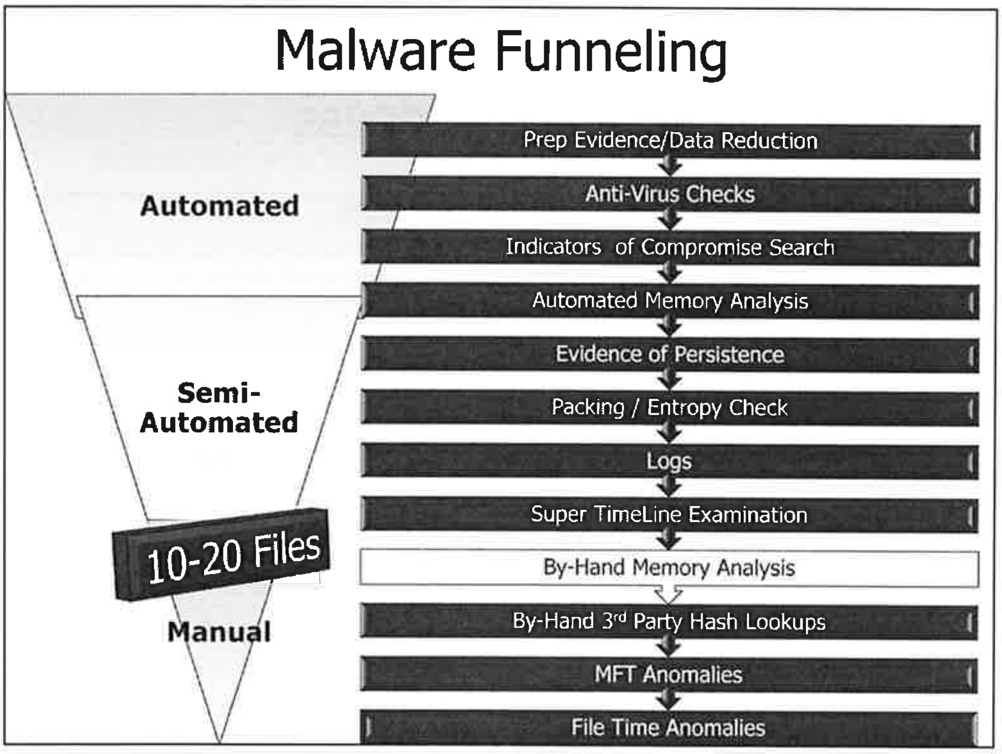


This page intentionally left blank.

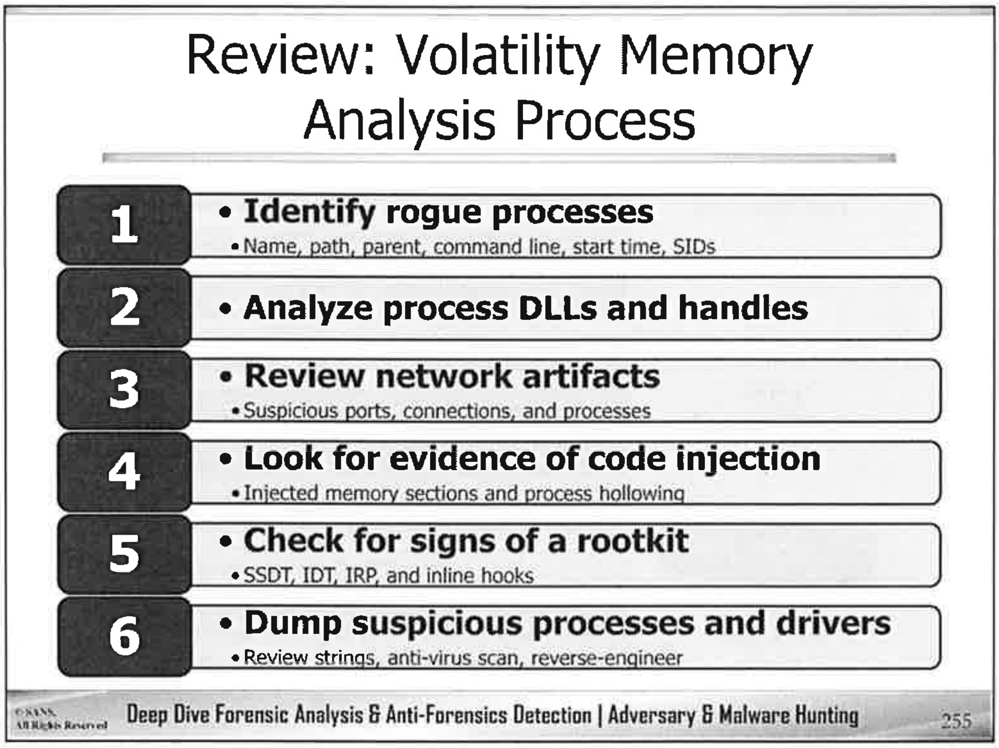


Once you are down to about 10-20 candidates at this point, it is always best to identify where those files show up in your timeline and examine the temporal proximity of the file itself. The context that could be discovered here could quickly eliminate many more files from being actual malware based on the use in the timeline itself.

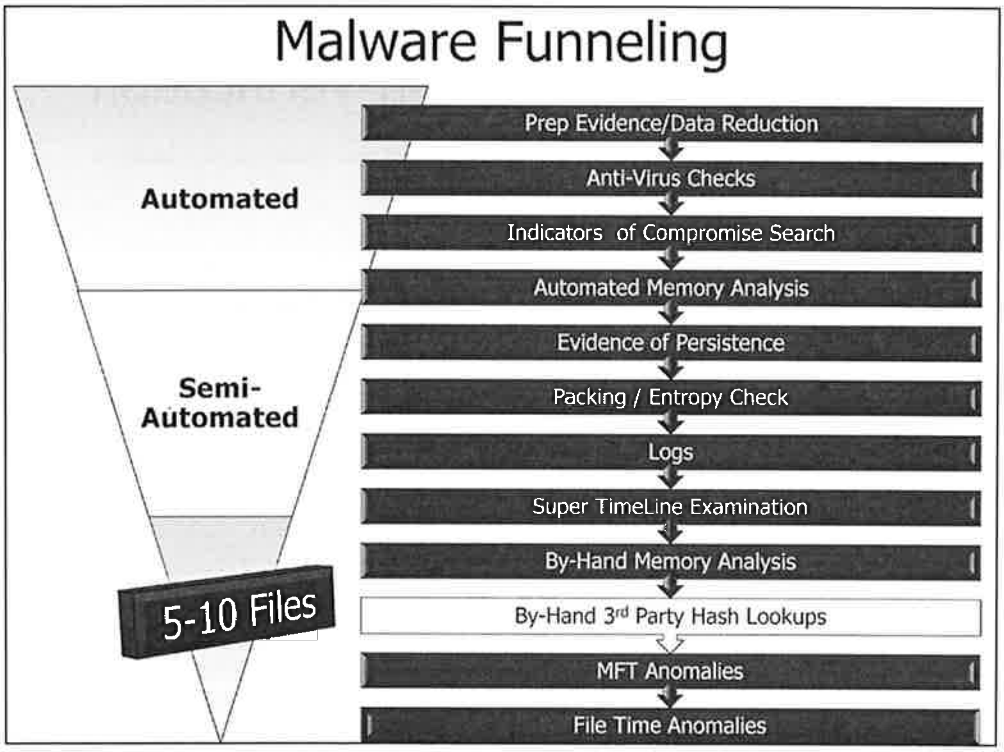
In contrast, in the above example, we can see the creation of the file winsvchost.exe in system32 directory. If this was one of your candidate files and you looked for when the file was created in the timeline, you would clearly see the traits that are seen during a spearphsing attack. Contextual clues in the temporal proximity of the files you are examining will be quite useful in your overall case.



This page intentionally left blank.



These steps are used when we utilize the Volatility memory analysis toolset. When using Volatility, our process will be the same, but we will be using a completely different toolkit. Both tools have their benefits and we recommend using both to check one another and provide the best of both worlds when doing your memory analysis.



This page intentionally left blank.

Bit9 FileAdvisor is the first free search engine for identifying software files on any Windows computer to help you achieve business-friendly whitelisting. Instead of relying on cryptic file names or unpredictable search engine results, FileAdvisor was built to make it easy for you to find exactly the information you are looking for. It is available free via its own web site http://fileadvisor.bit9.com, as well as via a downloadable utility (http://fileadvisor.bit9.com/services/wu/latest/FileAdvisor.msi).

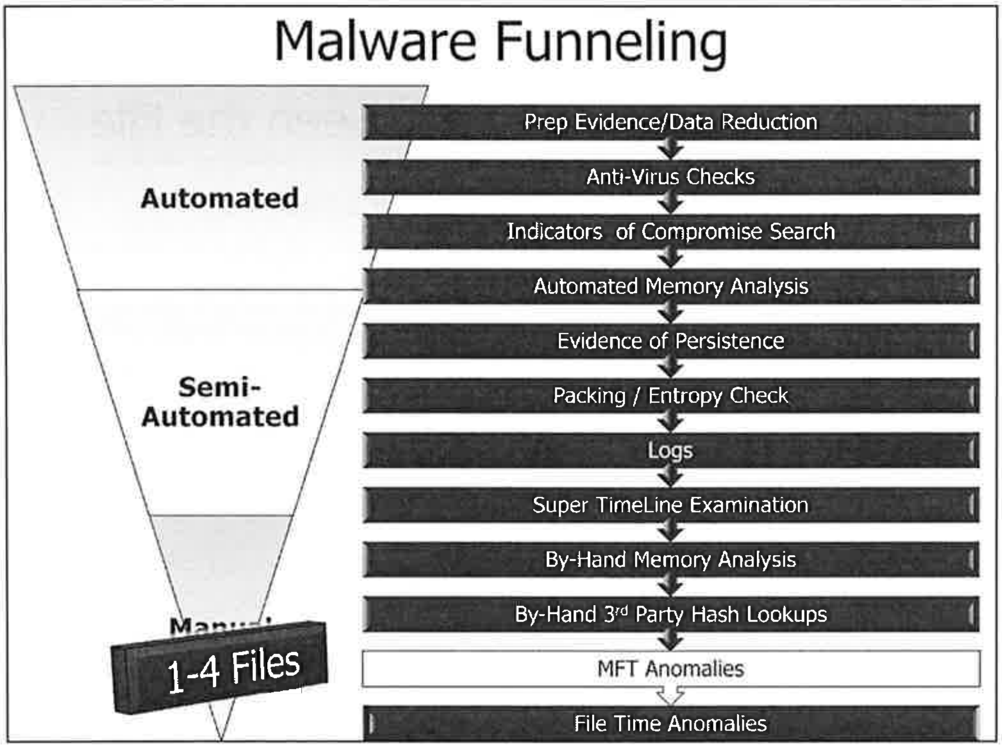
VirusTotal will scan a file through over 40 different A/V scanners to determine if any of the current signatures detect the malware. VirusTotal can also receive MD5 hashes of the possible malware candidates. One of the downsides of using Virustotal by submitting a file is that you will lose control of that malware and it might end up in A/V signatures later. If your goal is to keep your investigation as quiet as possible, many investigators opt to not submit their actual malware candidate files to the service as a result.

Bit9 File Advisor: See if Anyone Else has Seen the File

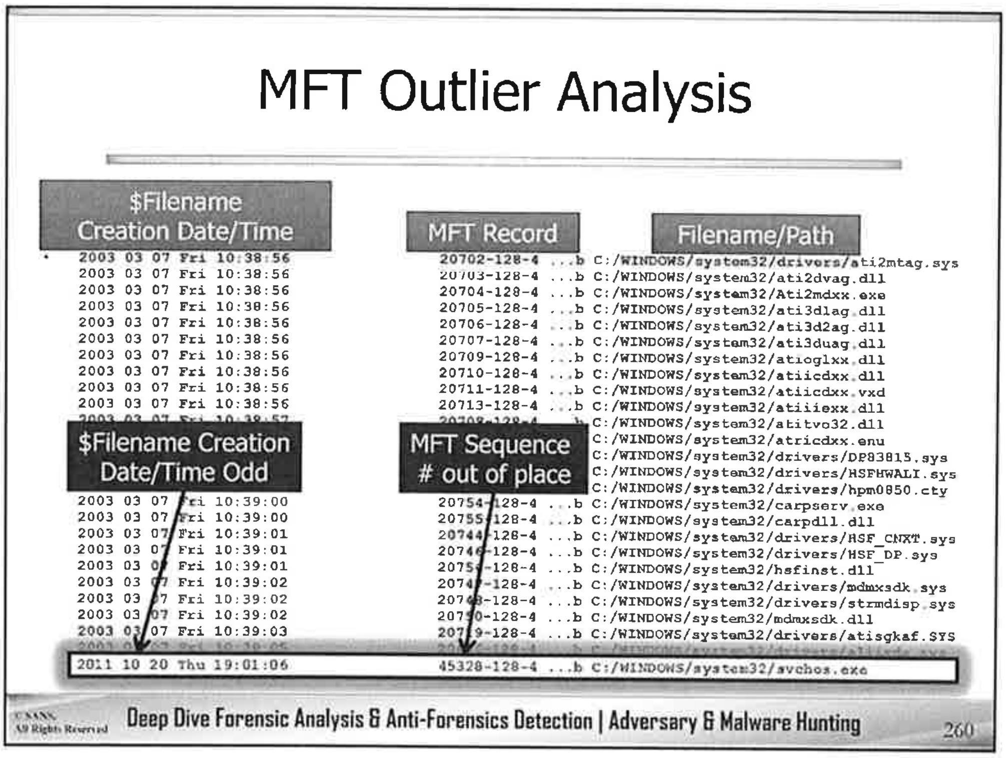
- http://fileadvisor.bit9.com Search files based on:
  - Filename
  - MD5 or SHA1 Hash

File Category	Description
Package	Group of associated files in which the file appeared. Depending on the file, packages can be nested in other packages or have parent packages. When possible, FileAdvisor displays this hierarchy as a tree. Files can appear in multiple packages.
Publisher	Publisher of the file as identified by digital certificate (if the file is signed).
Source	Name of the source where FileAdvisor discovered the file, which is not necessarily the publisher. To help you discern the nature and trustworthiness of the source, FileAdvisor further categorizes according to type (Internet, commercial software vendor, third-party provider, subscription, and open source project).

Bit9 has compiled the FileAdvisor database via an innovative, proprietary method of crawling the Web, through aggressive acquisition of physical media, as well as through partnerships with IBM and others. In addition, Bit9 FileAdvisor uses the National Institute of Standards and Technology's (NIST) National Software Reference Library (NSRL) Reference Data Set (RDS) -- also known as the NIST Special Database 28 -- as well as the Department of Justice's Hashkeeper database.



This page intentionally left blank.



On NTFS the first file on the file system is assigned MFT Record Number zero and is always the $MFT. As more files are created on the system, the next file on the system would get MFT Record Number one, and so on. On EXT2/3/4 file systems, the root directory (/), is assigned MFT Record Number two and things increase from there as additional files are stored on the system.

A typical file system has hundreds of thousands of files. Each file has its own MFT Record Number. Because of the way operating systems are installed, it's normal to see entire directory structures written to disk with files having largely sequential MFT Record Number values.

So MFT Record Numbers are a metadata structure akin to a card (dating myself here) from a library's card catalog. They contain information about the files in the same way that those cards used to contain author, title, number of pages, location in the library, etc., but MFT Record Numbers contain owner, group, location on disk, size of file, etc. In a library these cards are arranged alphabetically either by title, author or subject. In a file system, they are simply first come, first serve and they are numbered sequentially. In NTFS MFT Record Number 0 always points to the $MFT.

Given that these MFT Record Numbers are assigned sequentially, if new files are written to disk, the MFT Record Numbers that are assigned to them are likely to be sequential or close to sequential, assuming a sequential run of MFT Record Numbers is available. I need to hire a good illustrator to animate this concept.

Think of it this way, as files are deleted from the system, their MFT Record Numbers are marked as unallocated and are available for reuse. If there are no unallocated MFT Record Numbers, new ones will be assigned beginning with the current maximum MFT Record Number value plus one and so on.

A typical file system has hundreds of thousands of files. Each file has its own MFT Record Number. Because of the way operating systems are installed, it's normal to see entire directory structures written to disk with files having largely sequential MFT Record Number values, for example, here's a partial directory listing from a Windows NTFS partition's system32 directory (seen above)

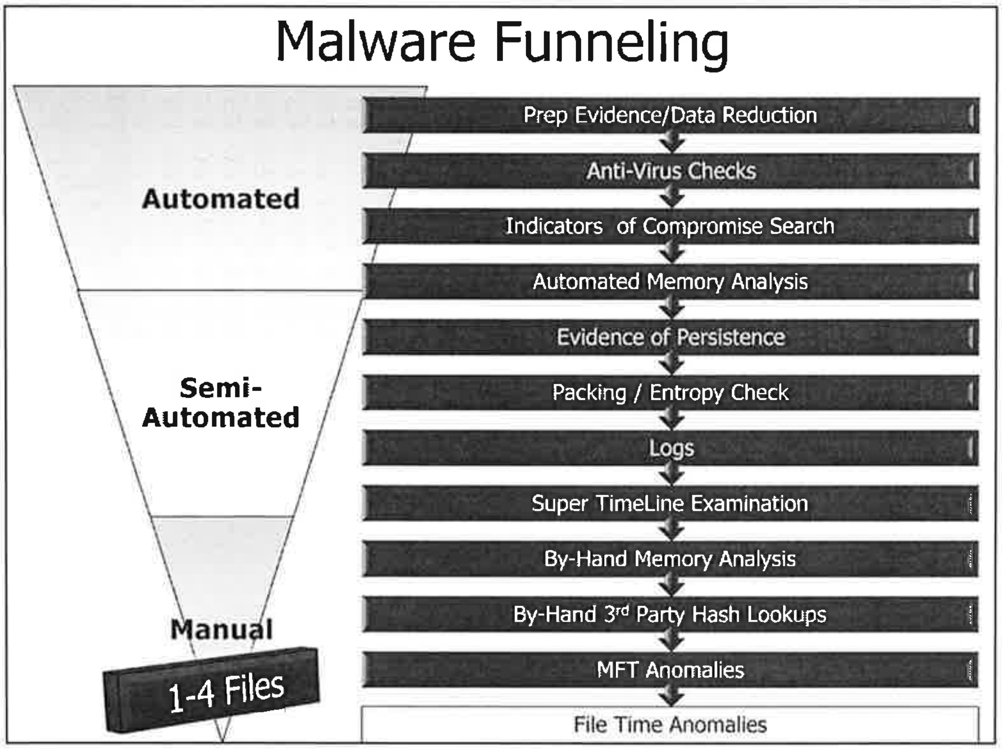
This partial listing is sorted by date, note the MFT Record Number values that follow the date stamps are largely sequential, there are some exceptions, but for the most part the order of MFT Record Number numbers aligns with the file's creation times. As file systems are used over the years and new patches are applied causing files to be backed up and replaced, the ordering of these files by MFT Record Number numbers breaks down, but surprisingly, this ordering remains intact enough on many systems, even after years of use, that we can use them to spot files that may be of interest.

You essentially look for outliers that are outside the creation date/time and have a MFT Record Number that is not in alignment with the others in that directory. The best way to accomplish this in many cases is to create a scatter plot of the directory itself. But this technique is not well known or widely used, but not many hackers have implemented in anti-forensics defense against this as they are not thinking about how the way the file system will perform MFT Record Number allocation.

$Filename Creation Date/Time	MFT Record		Filename/Path
2003 03 07 Fri 10:38:56	20702-128-4	...b	C:/WINDOWS/system32/drivers/ati2mtag.sys
2003 03 07 Fri 10:38:56	20703-128-4	...b	C:/WINDOWS/system32/ati2dvag.dll
2003 03 07 Fri 10:38:56	20704-128-4	...b	C:/WINDOWS/system32/Ati2mdxx.exe
2003 03 07 Fri 10:38:56	20705-128-4	...b	C:/WINDOWS/system32/ati3d1ag.dll
2003 03 07 Fri 10:38:56	20706-128-4	...b	C:/WINDOWS/system32/ati3d2ag.dll
2003 03 07 Fri 10:38:56	20707-128-4	...b	C:/WINDOWS/system32/ati3duag.dll
2003 03 07 Fri 10:38:56	20709-128-4	...b	C:/WINDOWS/system32/atioglxx.dll
2003 03 07 Fri 10:38:56	20710-128-4	...b	C:/WINDOWS/system32/atiicdxx.dll
2003 03 07 Fri 10:38:56	20711-128-4	...b	C:/WINDOWS/system32/atiicdxx.vxd
2003 03 07 Fri 10:38:56	20713-128-4	...b	C:/WINDOWS/system32/atiiiexx.dll
2003 03 07 Fri 10:38:57	20708-128-4	...b	C:/WINDOWS/system32/atitvo32.dll
			C:/WINDOWS/system32/atricdxx.enu
			C:/WINDOWS/system32/drivers/DP83815.sys
			C:/WINDOWS/system32/drivers/HSFHWALI.sys
			C:/WINDOWS/system32/drivers/hpm0850.cty
2003 03 07 Fri 10:39:00	20754-128-4	...b	C:/WINDOWS/system32/carpserv.exe
2003 03 07 Fri 10:39:00	20755-128-4	...b	C:/WINDOWS/system32/carpdll.dll
2003 03 07 Fri 10:39:01	20744-128-4	...b	C:/WINDOWS/system32/drivers/HSF_CNXT.sys
2003 03 07 Fri 10:39:01	20746-128-4	...b	C:/WINDOWS/system32/drivers/HSF_DP.sys
2003 03 07 Fri 10:39:01	2075[illegible]-128-4	...b	C:/WINDOWS/system32/hsfinst.dll
2003 03 07 Fri 10:39:02	2074[illegible]-128-4	...b	C:/WINDOWS/system32/drivers/mdmxsdk.sys
2003 03 07 Fri 10:39:02	20748-128-4	...b	C:/WINDOWS/system32/drivers/strmdisp.sys
2003 03 07 Fri 10:39:02	20750-128-4	...b	C:/WINDOWS/system32/mdmxsdk.dll
2003 03 07 Fri 10:39:03	207[illegible]9-128-4	...b	C:/WINDOWS/system32/drivers/atisgkaf.SYS
2011 10 20 Thu 19:01:06	45328-128-4	...b	C:/WINDOWS/system32/svchos.exe

$Filename Creation Date/Time Odd

MFT Sequence # out of place



This page intentionally left blank.

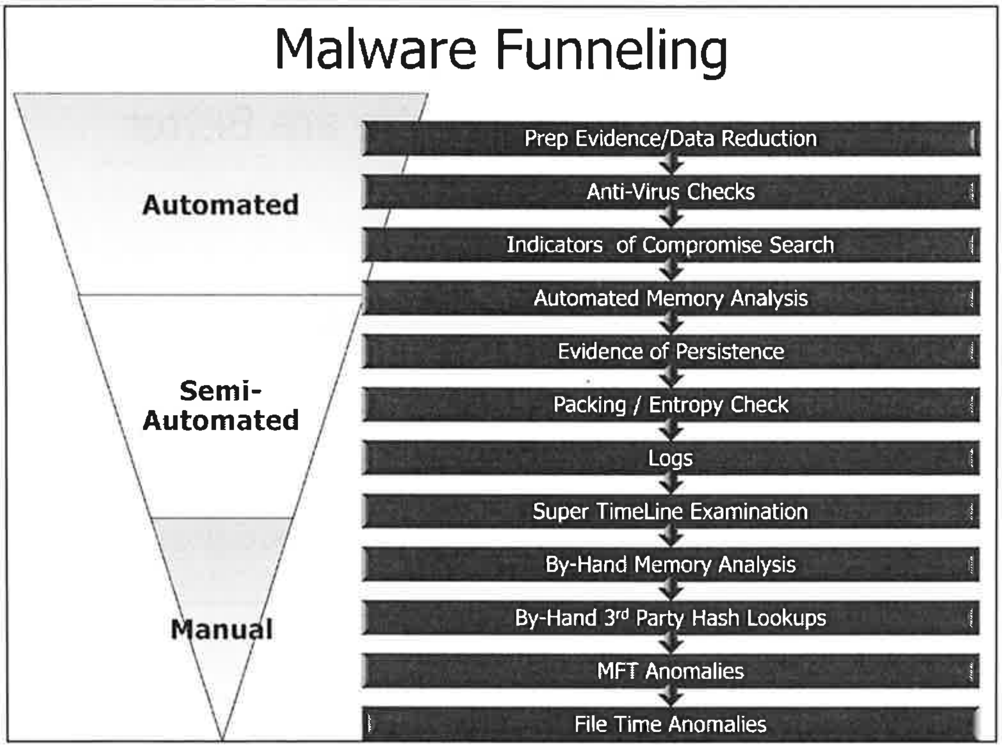
Timestamp Anomalies:
Hackers are Crafty: We are Better

- Timestamp Anomalies
  - $SI Time is before $FN Time
  - Nanoseconds values are all zeroes

H	I	M
Filename #1	Std Info Creation date	FN Info Creation date
winsvchost	8/12/2003 2:41	2/18/2007 20:41

One of the ways to tell if file time backdating occurred on a windows machine is to examine the filename times compared to the times stored in standard information. Tools such as timestomp allow a hacker to backdate a file to an arbitrary time of their choosing. Generally, hackers do this only to programs they are trying to hide in the system32 or similar system directories. Those directories and files would be a great place to start. You would look to see if the Filename (FN) time occurs after the Standard Info Creation Time as this would indicate an anomaly.

You can also compare this to the program compile time to see if the program was compiled after the creation date locally on the machine using the EXIFTool talked about in FOR408.

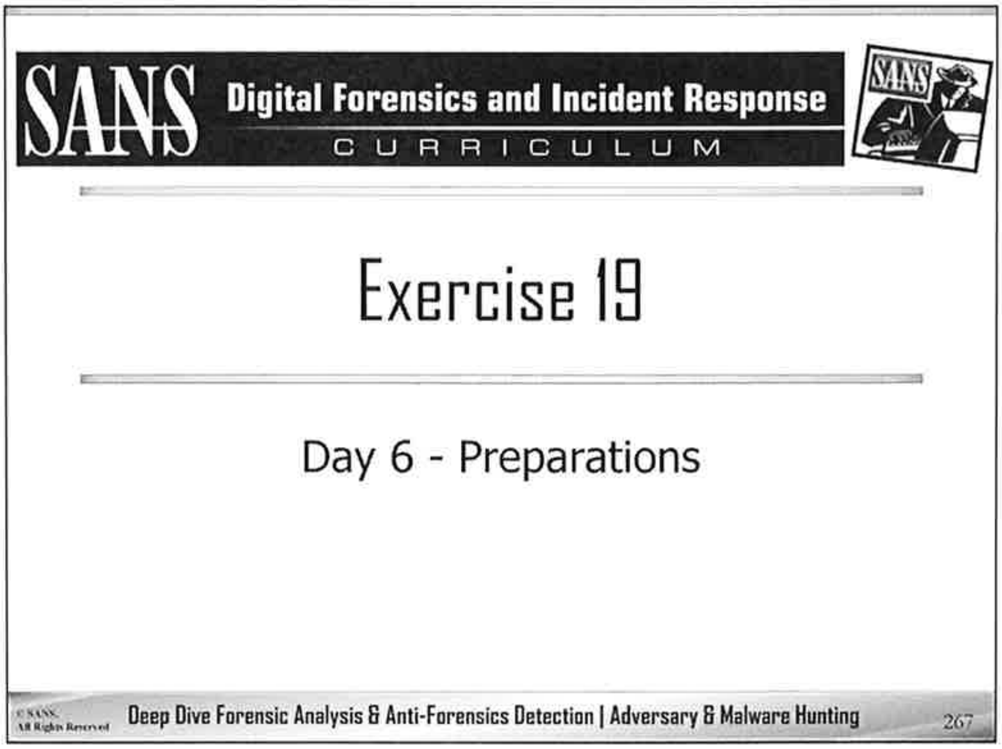


This page intentionally left blank.

You Have Malware! Now What?

- Hand it to Reverse Engineer
  - FOR610 – RE Malware
- Output from RE Engineer
  - Host based indicators
  - Network based indicators
- You can now find additional systems compromised by the malware you found

This page intentionally left blank.



This page intentionally left blank.

Filesystem Forensics and Intrusion Analysis

WinXP Restore Point Analysis

Volume Shadow Analysis

Deep Dive Forensics

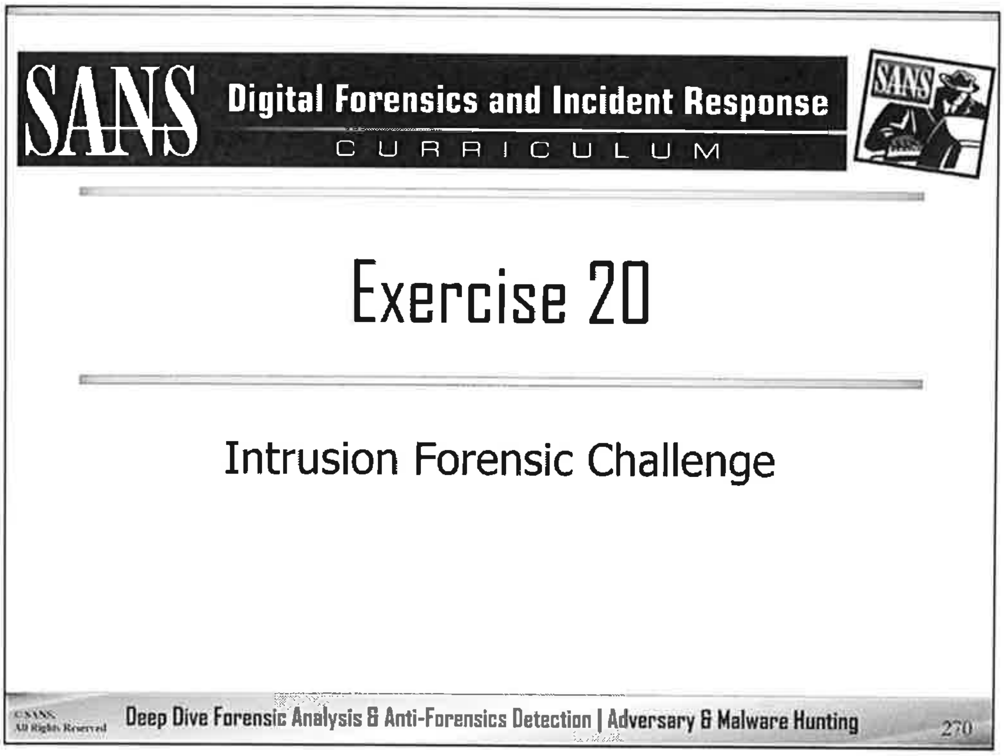
- Filesystem Based Analysis
- Sleuthkit Toolset
- Partition / Volume Analysis
- Data Layer Analysis
- Stream-Based Data Carving
- File-Based Data Carving
- NTFS Filesystem Analysis
- FAT/exFAT Filesystem Overview

Art of Finding Unknown Malware

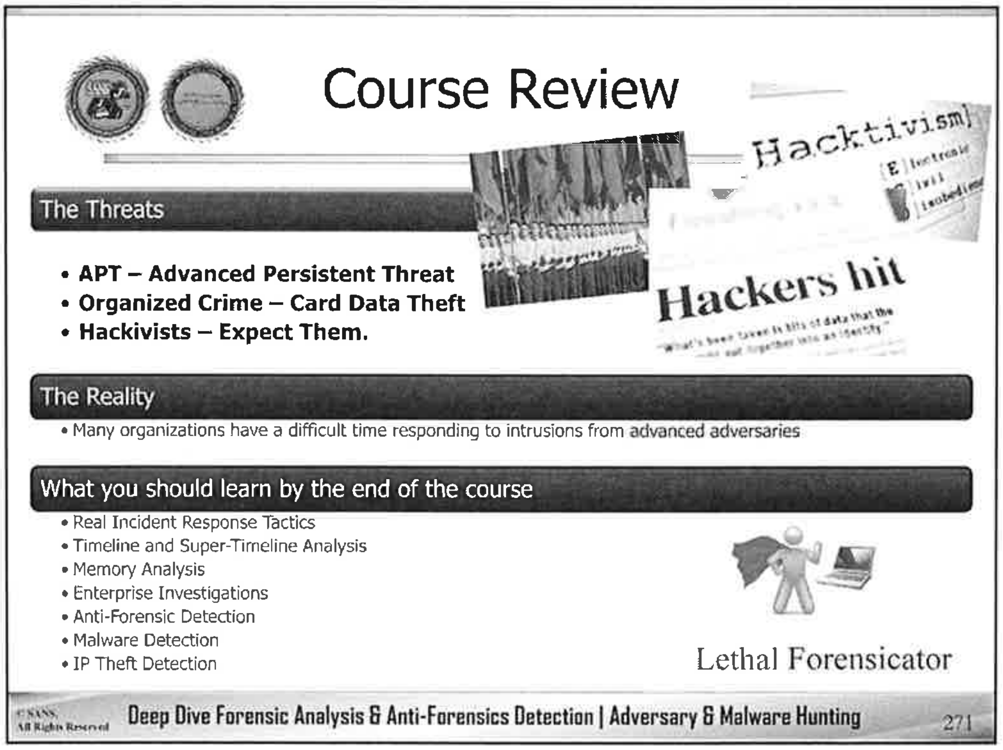
Investigating Hackers or Technically Savvy Criminals

This page intentionally left blank.

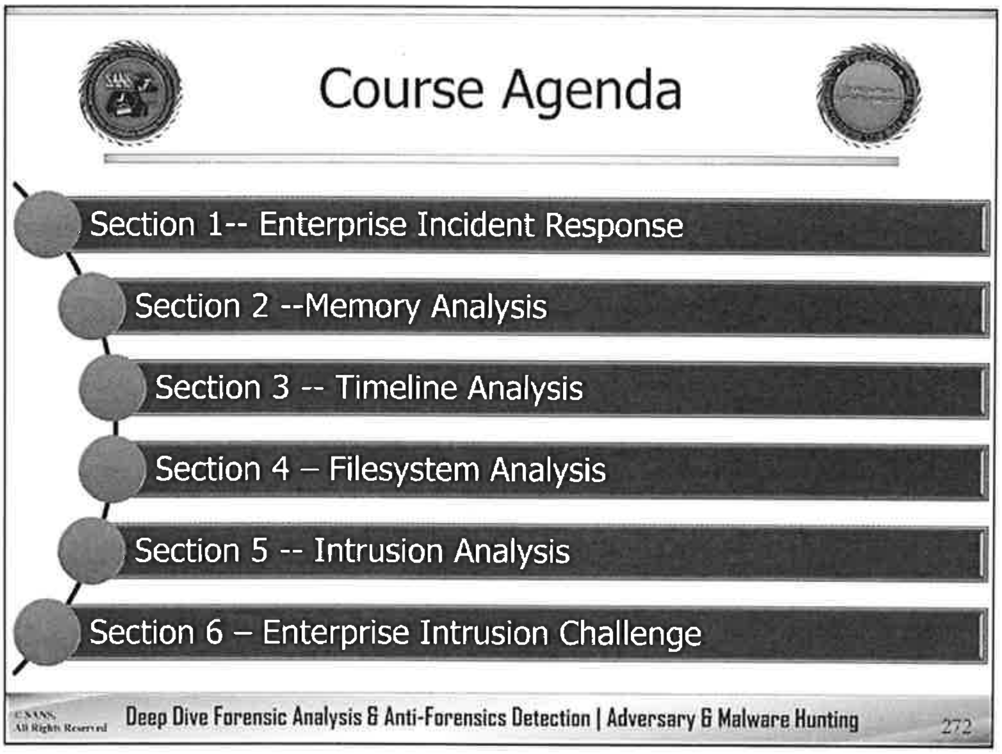
Author: Rob Lee
rlee@sans.org
http://twitter.com/robtlee
http://twitter.com/sansforensics



This page intentionally left blank.



This page intentionally left blank.



This page intentionally left blank.

This page intentionally left blank.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

Website
digital-forensics.sans.org

SIFT Workstation
dfir.to/SANS-SIFT

Join The SANS DFIR Community

Blog: dfir.to/DFIRBlog

Twitter: @sansforensics

Facebook: sansforensics

Google+: gplus.to/sansforensics

Mailing list: dfir.to/MAIL-LIST

YouTube: dfir.to/DFIRCast

DFIR CURRICULUM

CORE

FOR408
Windows Forensics
GCFE

SEC504
Hacker Tools, Techniques, Exploits & Incident Handling
GCIH

OPERATING SYSTEM & DEVICE IN-DEPTH

FOR585
Advanced Smartphone Forensics

FOR518
Mac Forensics

FOR526
Memory Forensics In-Depth

INCIDENT RESPONSE & ADVERSARY HUNTING

FOR508
Advanced Incident Response
GCFA

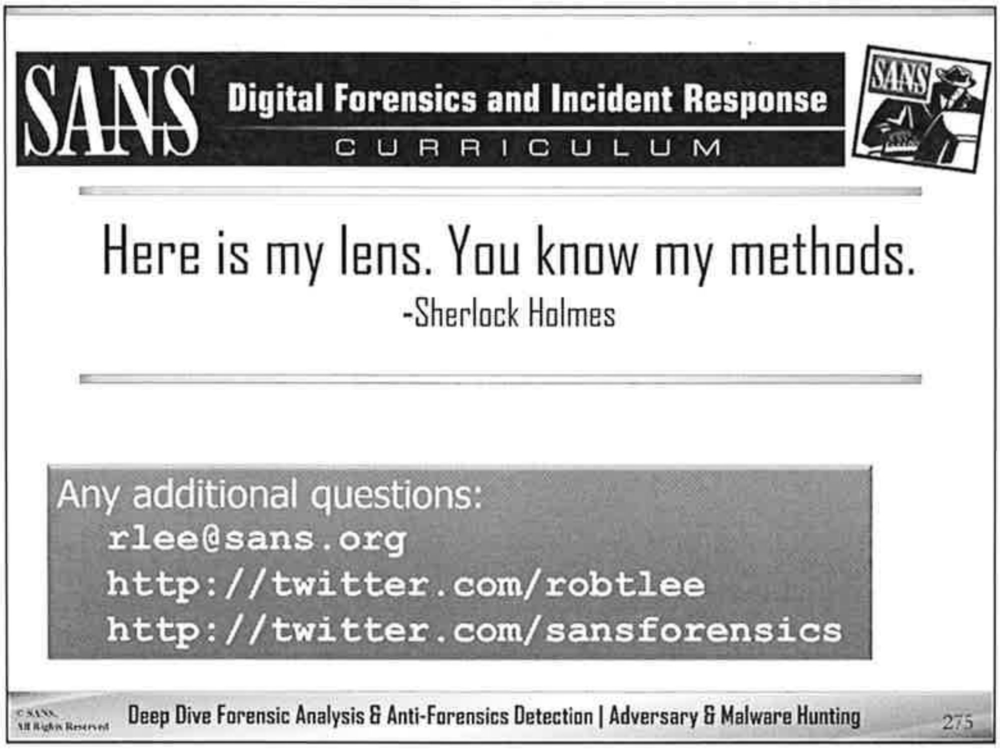
FOR572
Advanced Network Forensics and Analysis
GNFA

FOR578
Cyber Threat Intelligence

LEARN REM

FOR610
REM: Malware Analysis
GREM

MGT535
Incident Response Team Management



This page intentionally left blank.

This page intentionally left blank.

SANS Digital Forensics and Incident Response
CURRICULUM

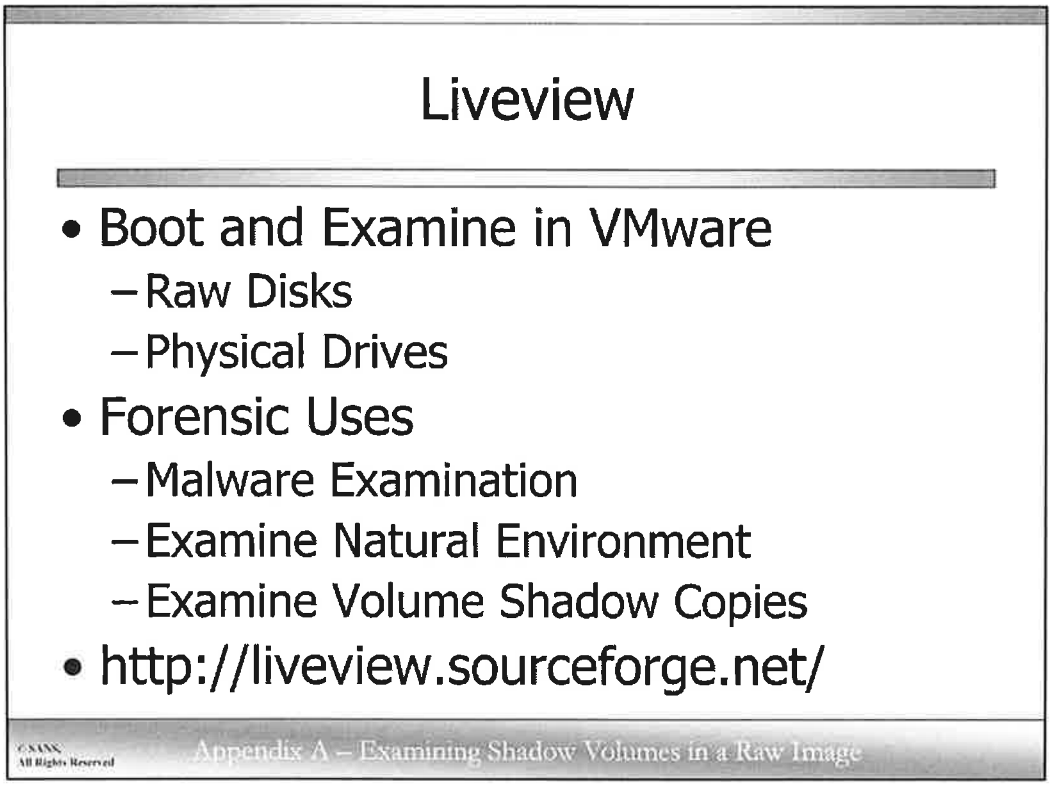
Examining Shadows Volumes in a Raw Image

This page intentionally left blank.

Image Raw Physical Drive

```
D:\IR\FAU>dd.exe if=\\.\PhysicalDrive1 of=E:\win7_image.001 --localwrt
The VistaFirewall Firewall is active with exceptions.


Copying \\.\PhysicalDrive1 to E:\win7_image.001
Output: E:\win7_image.001
200049647616 bytes
190782+1 records in
190782+1 records out
200049647616 bytes written

Succeeded!

D:\IR\FAU>
```


This page intentionally left blank.

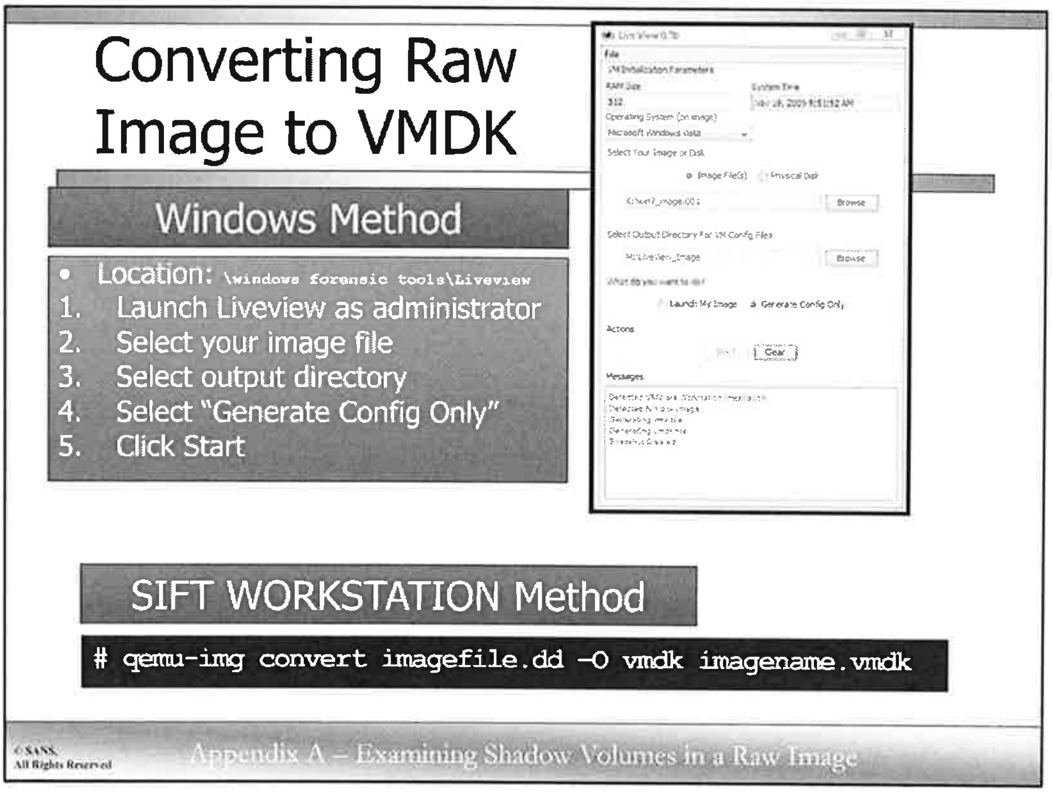


From the Sourceforge site: Live View is a Java-based graphical forensics tool that creates a VMware virtual machine out of a raw (dd-style) disk image or physical disk. This allows the forensic examiner to "boot up" the image or disk and gain an interactive, user-level perspective of the environment, all without modifying the underlying image or disk. Because all changes made to the disk are written to a separate file, the examiner can instantly revert all of his or her changes back to the original pristine state of the disk. The end result is that one need not create extra "throw away" copies of the disk or image to create the virtual machine. Live View is developed by CERT, Software Engineering Institute.

I only use Liveview for two reasons. 1. If I ever need to boot into the machine I'm examining to see how a piece of malware works or 2. Examining Shadow Volumes.

If a piece of malware needs to be examined in the original environment it is sometimes useful to boot the machine into vmware to be able to examine it.

Using Liveview to create a bootable version of a raw disk is easy work and quiet fast as well.

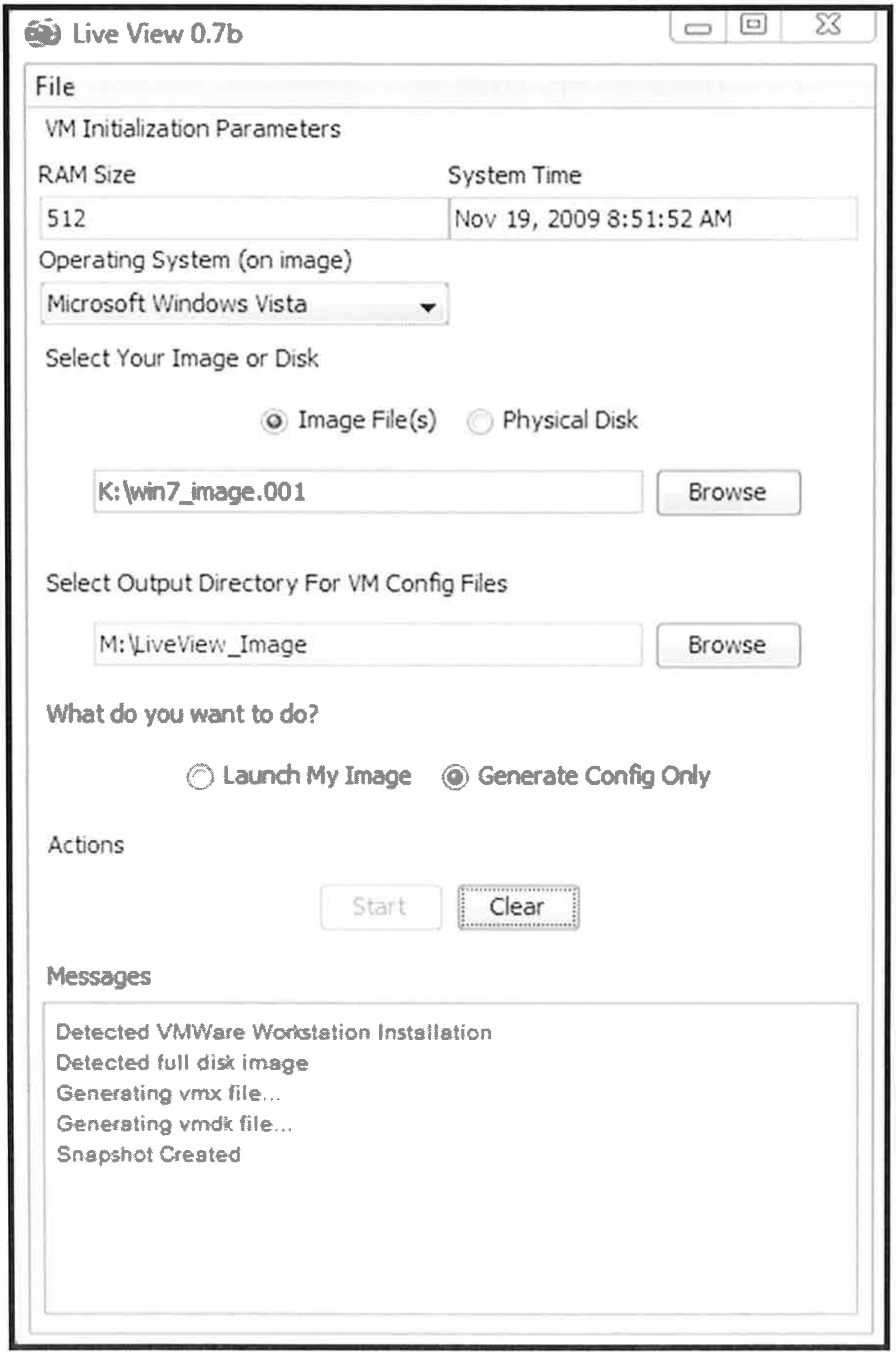


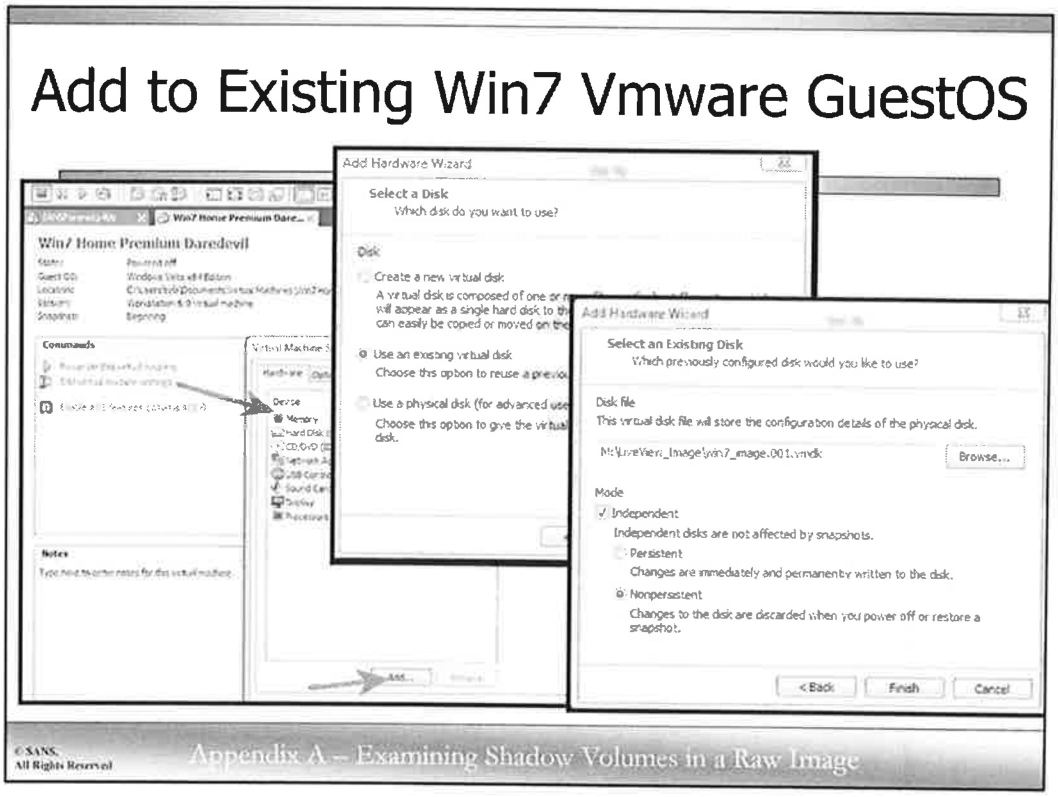
Location: `\windows forensic tools\Liveview`

1. Launch Liveview as administrator
2. Select your image file
3. Select output directory
4. Select "Generate Config Only"
5. Click Start

Alternatively you could use the utility qemu-img inside the SIFT workstation to convert the image.

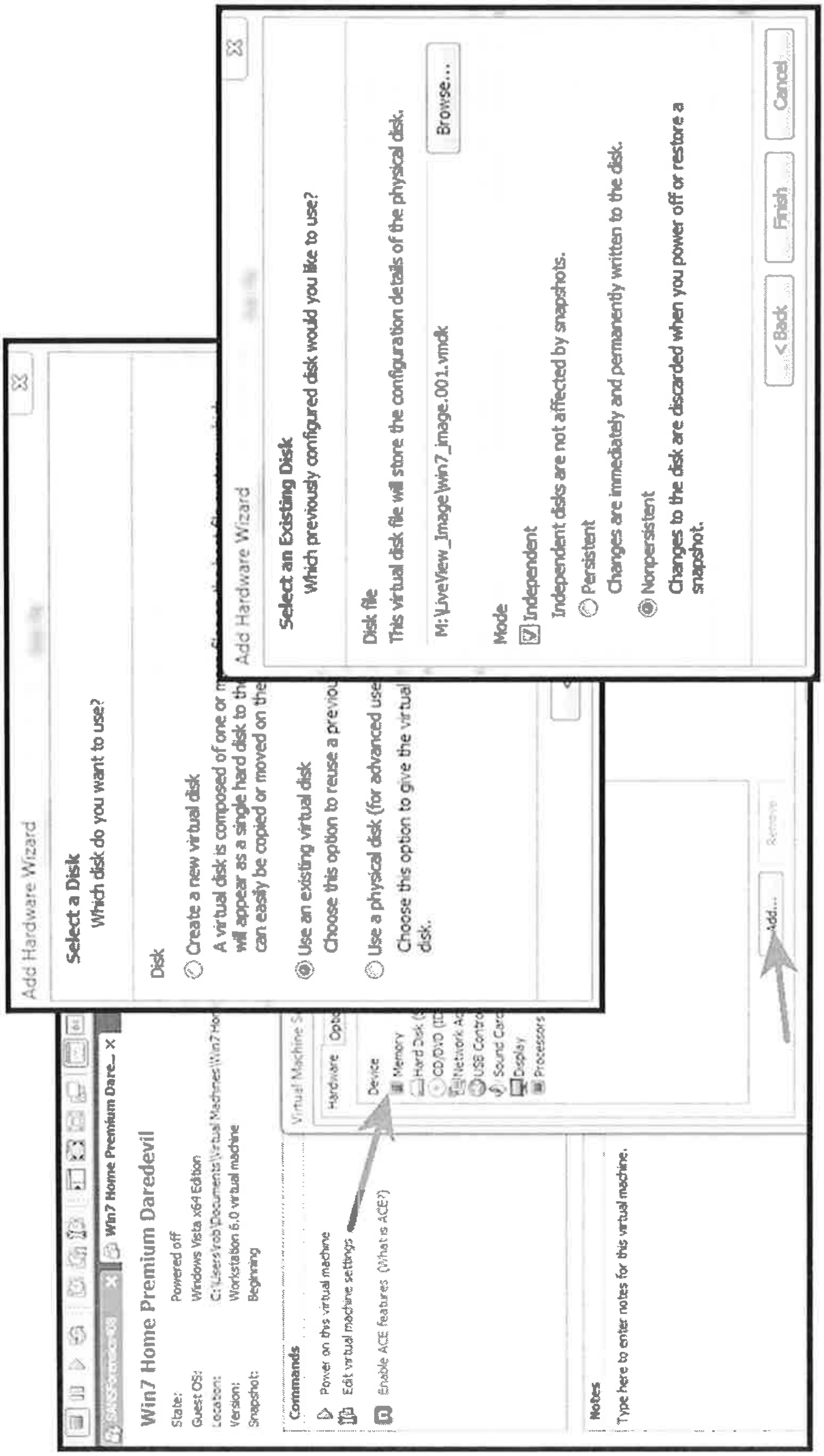
qemu-img convert imagefile.dd –O vmdk imagename.vmdk


Live View 0.7b
File
VM Initialization Parameters
RAM Size
System Time
512
Nov 19, 2009 8:51:52 AM
Operating System (on image)
Microsoft Windows Vista
Select Your Image or Disk
Image File(s)
Physical Disk
K:\win7_image.001
Browse
Select Output Directory For VM Config Files
M:\LiveView_Image
Browse
What do you want to do?
Launch My Image
Generate Config Only
Actions
Start
Clear
Messages
Detected VMWare Workstation Installation
Detected full disk image
Generating vmx file...
Generating vmdk file...
Snapshot Created

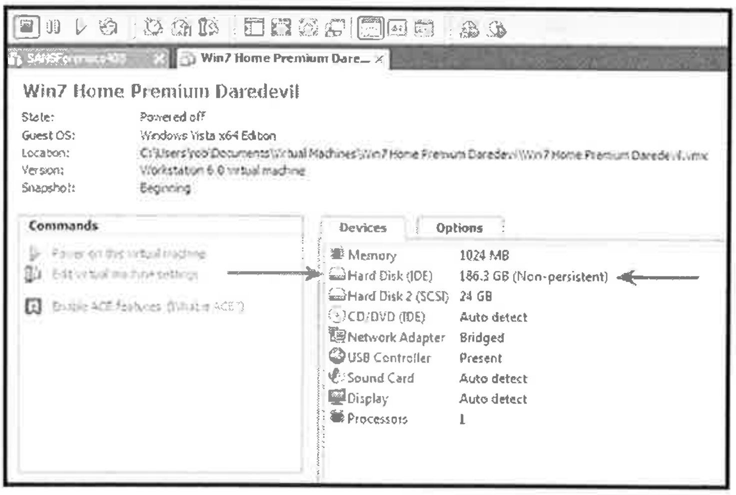


Add To Existing Win7 Vmware GuestOS

1. Open Windows 7 Virtual Machine.
2. Click Edit Virtual Machine Settings.
3. Click Add -> Hard Disk -> Next.
4. Select "Use an existing virtual disk".
5. Select the location for the vmdk file that was just created from Liveview.
6. Select the disk to be Independent and Non-persistent. The reason why we set it to be Non-persistent is to make it a repeatable and reproducible process which is more forensically sound.

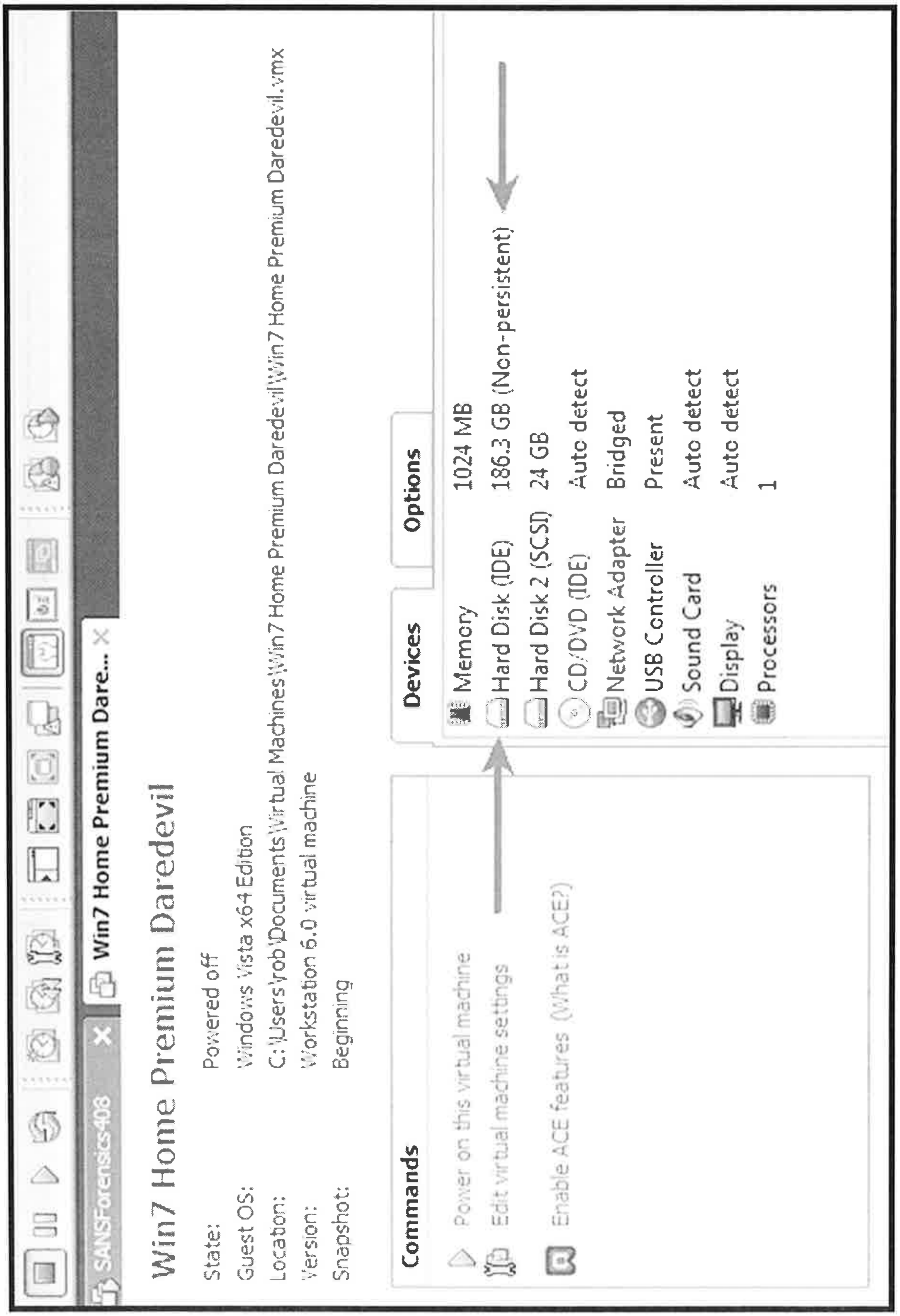

Win7 Home Premium Daredevil
State: Powered off
Guest OS: Windows Vista x64 Edition
Version: Workstation 6.0 virtual machine
Snapshot: Beginning
Commands
Power on this virtual machine
Edit virtual machine settings
Enable ACE features (What is ACE?)
Notes
Type here to enter notes for this virtual machine.
Hardware
Device
Memory
Processors
Add...
Remove
Add Hardware Wizard
Select a Disk
Which disk do you want to use?
Disk
Create a new virtual disk
Use an existing virtual disk
Choose this option to reuse a previou
Use a physical disk (for advanced use
Choose this option to give the virtual
disk.
Add Hardware Wizard
Select an Existing Disk
Which previously configured disk would you like to use?
Disk file
This virtual disk file will store the configuration details of the physical disk.
M:\LiveView_Image\win7_image.001.vmdk
Browse...
Mode
Independent
Independent disks are not affected by snapshots.
Persistent
Changes are immediately and permanently written to the disk.
Nonpersistent
Changes to the disk are discarded when you power off or restore a snapshot.
< Back
Finish
Cancel

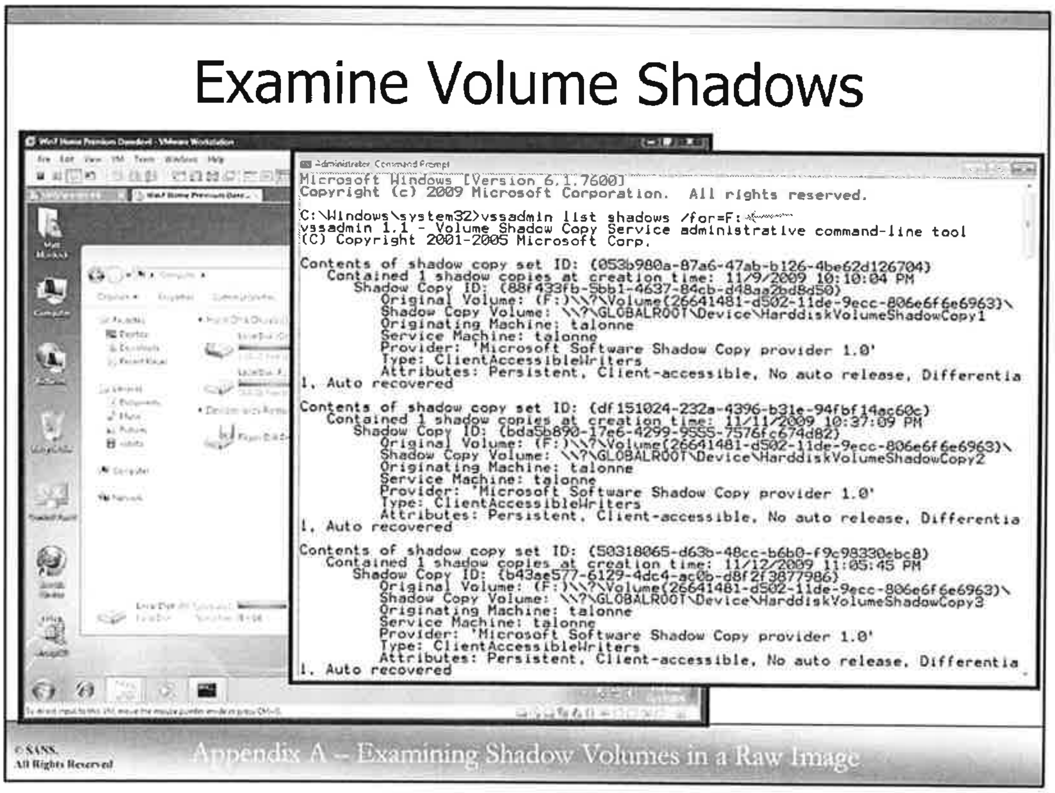
Power on Virtual Machine



Appendix A – Examining Shadow Volumes in a Raw Image

Power on the virtual machine with the non-persistent hard drive that contains the shadows you wish to examine.


SANSForensics408
Win7 Home Premium Dare...
Win7 Home Premium Daredevil
State: Powered off
Guest OS: Windows Vista x64 Edition
Location: C:\Users\rob\Documents\Virtual Machines\Win7 Home Premium Daredevil\Win7 Home Premium Daredevil.vmx
Version: Workstation 6.0 virtual machine
Snapshot: Beginning
Commands
Power on this virtual machine
Edit virtual machine settings
Enable ACE features (What is ACE?)
Devices
Options
Memory 1024 MB
Hard Disk (IDE) 186.3 GB (Non-persistent)
Hard Disk 2 (SCSI) 24 GB
CD/DVD (IDE) Auto detect
Network Adapter Bridged
USB Controller Present
Sound Card Auto detect
Display Auto detect
Processors 1

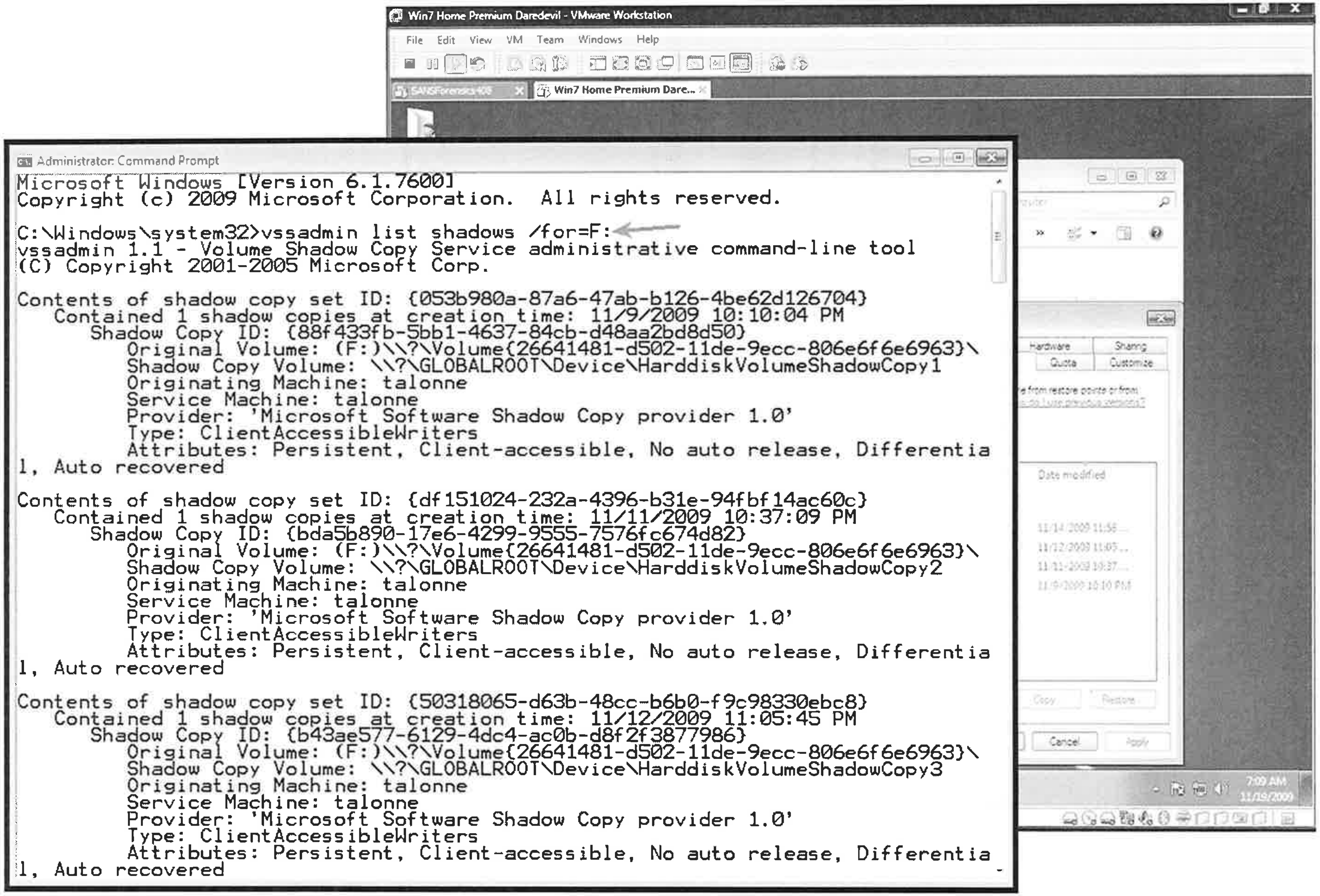


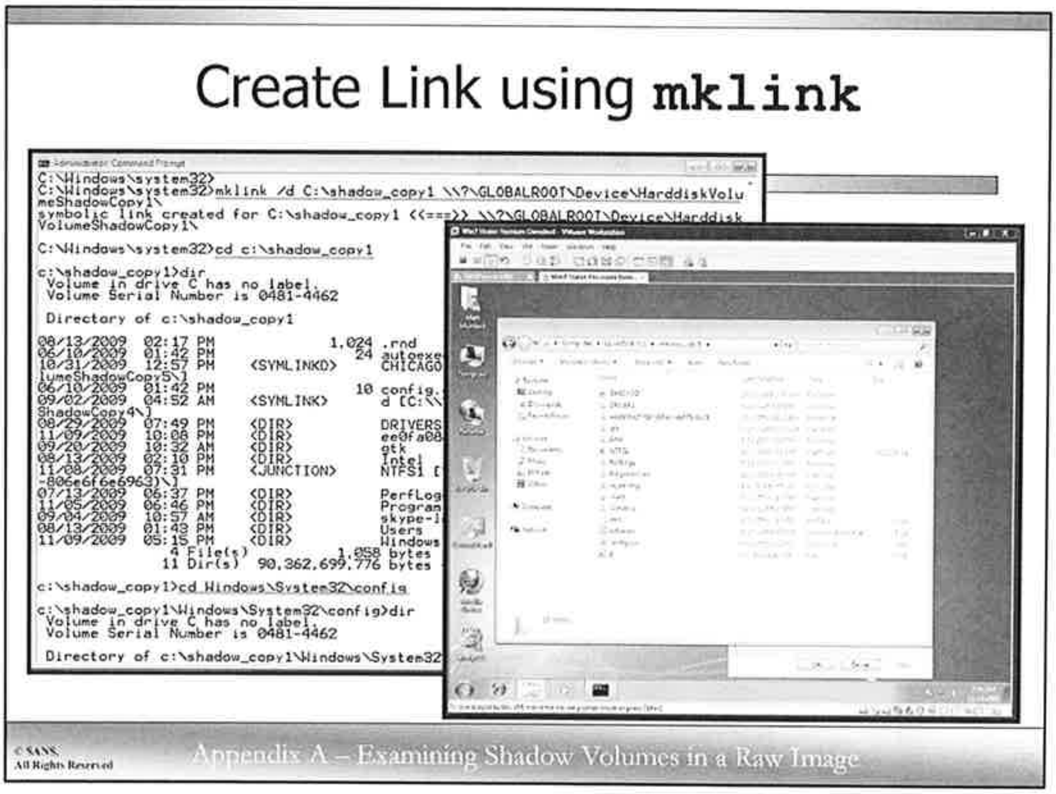
Your new vmdk will show up as a drive letter. In most cases with Win7 you should see two partitions now. The second drive should be your C:\ drive that is where you will find your Volume Shadows.

First off you can easily see the Volume Shadows by right clicking on the "Restore Previous Versions" when you right click on the Local Disk (F:).

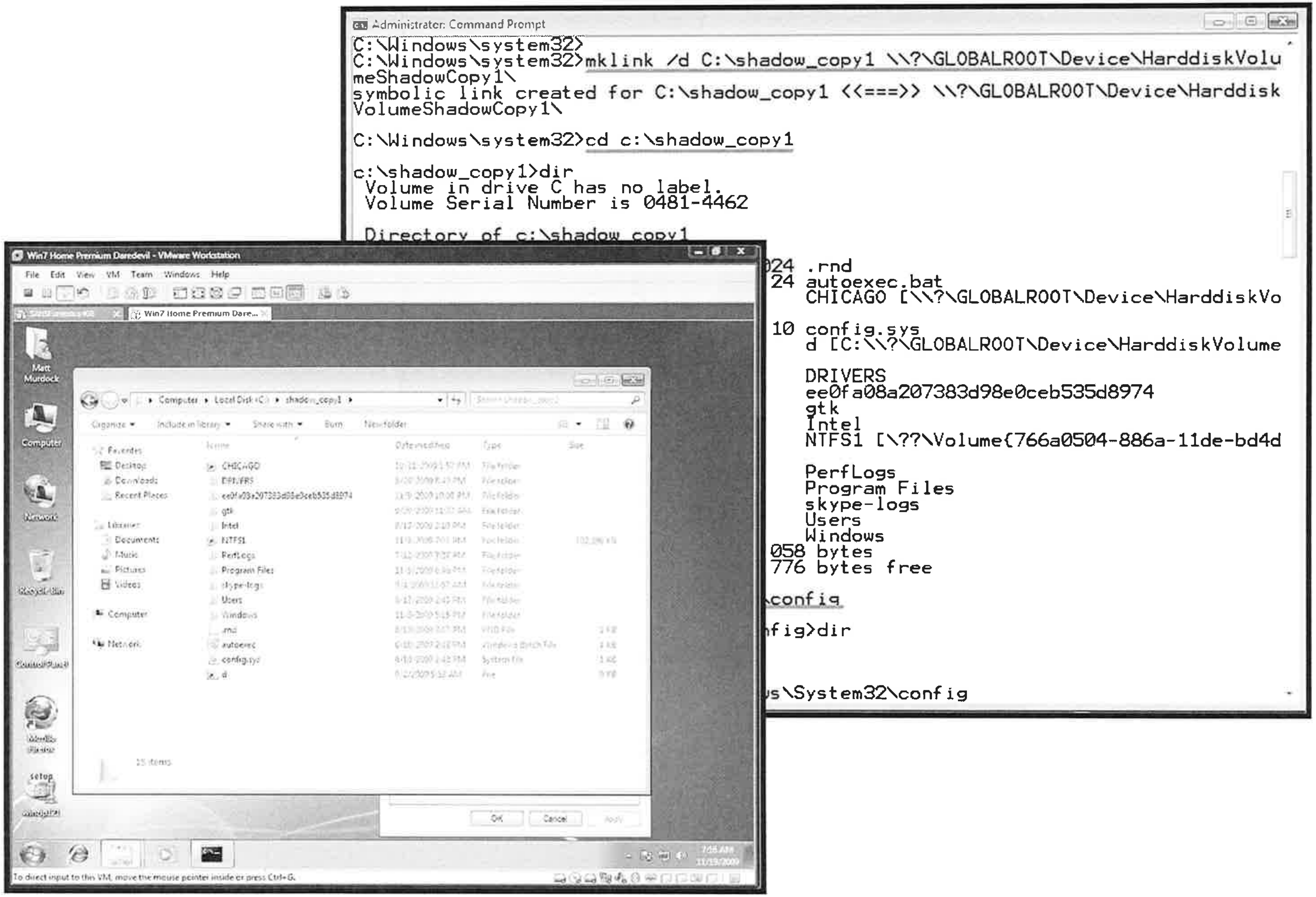
Start an administrator enabled command prompt and list shadows.

```
C:\vssadmin list shadows /for=F:
```


Win7 Home Premium Daredevil - VMware Workstation
File Edit View VM Team Windows Help
Win7 Home Premium Dare...
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>vssadmin list shadows /for=F:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.
Contents of shadow copy set ID: {053b980a-87a6-47ab-b126-4be62d126704}
Contained 1 shadow copies at creation time: 11/9/2009 10:10:04 PM
Shadow Copy ID: {88f433fb-5bb1-4637-84cb-d48aa2bd8d50}
Original Volume: (F:)\\?\Volume{26641481-d502-11de-9ecc-806e6f6e6963}\
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
Originating Machine: talonne
Service Machine: talonne
Provider: 'Microsoft Software Shadow Copy provider 1.0'
Type: ClientAccessibleWriters
Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered
Contents of shadow copy set ID: {df151024-232a-4396-b31e-94fbf14ac60c}
Contained 1 shadow copies at creation time: 11/11/2009 10:37:09 PM
Shadow Copy ID: {bda5b890-17e6-4299-9555-7576fc674d82}
Original Volume: (F:)\\?\Volume{26641481-d502-11de-9ecc-806e6f6e6963}\
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
Originating Machine: talonne
Service Machine: talonne
Provider: 'Microsoft Software Shadow Copy provider 1.0'
Type: ClientAccessibleWriters
Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered
Contents of shadow copy set ID: {50318065-d63b-48cc-b6b0-f9c98330ebc8}
Contained 1 shadow copies at creation time: 11/12/2009 11:05:45 PM
Shadow Copy ID: {b43ae577-6129-4dc4-ac0b-d8f2f3877986}
Original Volume: (F:)\\?\Volume{26641481-d502-11de-9ecc-806e6f6e6963}\
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy3
Originating Machine: talonne
Service Machine: talonne
Provider: 'Microsoft Software Shadow Copy provider 1.0'
Type: ClientAccessibleWriters
Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered
7:09 AM
11/19/2009




```
C:\>  mklink /d
C:\shadow_copy1\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\
```


Administrator: Command Prompt
C:\Windows\system32>
C:\Windows\system32>mklink /d C:\shadow_copy1 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\
symbolic link created for C:\shadow_copy1 <<===>> \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\
C:\Windows\system32>cd c:\shadow_copy1
c:\shadow_copy1>dir
Volume in drive C has no label.
Volume Serial Number is 0481-4462
Directory of c:\shadow_copy1
.rnd
autoexec.bat
CHICAGO [\\?\GLOBALROOT\Device\HarddiskVo
config.sys
d [C:\\?\GLOBALROOT\Device\HarddiskVolume
DRIVERS
ee0fa08a207383d98e0ceb535d8974
gtk
Intel
NTFS1 [\??\Volume{766a0504-886a-11de-bd4d
PerfLogs
Program Files
skype-logs
Users
Windows
bytes
bytes free
fig>dir
s\System32\config
Win7 Home Premium Daredevil - VMware Workstation
File Edit View VM Team Windows Help
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Imaging Shadow Volume

```
Administrator: Command Prompt - dd if=\\.\HarddiskVolumeShadowCopy1 of=G:\Shadow_Image\Win7_ShadowCopy1.001 --localwrt

D:\IR\FAU>dd if=\\.\HarddiskVolumeShadowCopy1 of=G:\Shadow_Image\Win7_ShadowCopy
1.001  --localwrt
The VistaFirewall Firewall is active with exceptions.

Copying \\.\HarddiskVolumeShadowCopy1 to G:\Shadow_Image\Win7_ShadowCopy1.001
```


Finally, you can image the shadow in the Vmware host using George Garner's dd.exe.

```
Administrator: Command Prompt - dd if=\\.\HarddiskVolumeShadowCopy1 of=G:\Shadow_Image\Win7_ShadowCopy1.001 --localwrt

D:\IR\FAU>dd if=\\.\HarddiskVolumeShadowCopy1 of=G:\Shadow_Image\Win7_ShadowCopy
1.001  --localwrt
The VistaFirewall Firewall is active with exceptions.

Copying \\.\HarddiskVolumeShadowCopy1 to G:\Shadow_Image\Win7_ShadowCopy1.001
```

SANS Digital Forensics and Incident Response CURRICULUM

Shadow Timelines

This page intentionally left blank.

Vista/Windows 7: Shadow Timelines

- Volume Shadow Copy
  - Exact copy of filesystem
  - Early date/time
  - MACB times and files intact
  - What was gone today could be there yesterday
- Requires **fls.exe** for Windows systems
  - C:\windows forensic tools\sleuthkit-windows\fls.exe
- Output to a shared network drive on SIFTWORKSTATION

Since you have multiple copies of the same volume via a shadow volume on the same hard drive. You can now extract data from previous write times to the same files. This exciting new possibility shows the last time files were modified. Not just the last modified time, but all of the last modified times stored in the VSC.

In 2010, I was upgrading the SIFT Workstation to the new version and I realized I had not used the Windows version of the Sleuthkit tools in awhile. I usually demonstrate in class that many of the sleuthkit tools can work directly against the logical partitions of a Physical Hard Drive (e.g. \\.\C:, \\.\D:). It occurred to me that I had never tried to use the filesystem parser and timeline generator **fls** on a Windows Vista, Windows 7, or Windows 2008 Server ShadowCopyVolume. (Note: for some odd reason I have had difficulty getting fls to work on a x64 based systems. Some sleuthkit commands work while others do not.)

We have known for some time now that you can image a Shadow Volume. I wrote a post back in 2008 describing it titled VISTA and Windows 7 Shadow Volume Forensics. We found out you could use dd.exe to carve a logical volume at \\.\C: We also found out we could use the same command to point at a ShadowCopy called \\.\HarddiskVolumeShadowCopy#. However, after that entire year, I never once attempted to run any of the sleuthkit tools against the logical volume of a shadow copy volume.

The potential for full forensics could be significant. However, with everything else, I always begin with the timeline to see if the basics would work. It did. Here is what I found out.

I launched an administrator enabled command prompt in the sleuthkit-windows directory from my CDROM. Again, I want to mimic that someone might want to run this during incident response. You can easily convert a dd image into a VMDK using liveview and load into a Windows 7 workstation virtual machine in non-persistent mode if you want to examine shadow volumes from a raw/dd image as well.

Which Shadow Will You Use?

```
D:\winexercise\windows forensic tools\sleuthkit-windows>vssadmin list shadows
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Contents of shadow copy set ID: {5ba11162-afea-4640-be89-dd3523cca3bf}
   Contained 1 shadow copies at creation time: 7/16/2009 6:31:58 PM
      Shadow Copy ID: {87c53b99-f678-4ece-976f-cc072ad4fc34}
         Original Volume: (C:)\\?\Volume{f87e43cb-3ff1-11de-9b80-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
         Originating Machine: WIN-V5MTNQS4ECM
         Service Machine: WIN-V5MTNQS4ECM
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
l, Auto recovered

Contents of shadow copy set ID: {0a4d54a6-0fb5-4104-bb2e-67db5398be42}
   Contained 1 shadow copies at creation time: 8/13/2009 8:06:44 PM
      Shadow Copy ID: {dd68f268-6ca3-44dc-a089-3af24c3a9a6f}
         Original Volume: (C:)\\?\Volume{f87e43cb-3ff1-11de-9b80-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
         Originating Machine: WIN-V5MTNQS4ECM
         Service Machine: WIN-V5MTNQS4ECM
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
l, Auto recovered
```


D:\windows forensic tools\sleuthkit-windows> fls -r -m C: \\.\C: >
\\SIFTWORKSTATION\cases\ShadowTime\bodyfile

Again you point to the shadow volume you are interested in obtaining the data from to add to your bodyfile.

We start with the command to extract our current filesystem data and send it across the network to our waiting SIFT Workstation 2.0.

```
D:\winexercise\windows forensic tools\sleuthkit-windows>vssadmin list shadows
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Contents of shadow copy set ID: {5ba11162-afea-4640-be89-dd3523cca3bf}
   Contained 1 shadow copies at creation time: 7/16/2009 6:31:58 PM
      Shadow Copy ID: {87c53b99-f678-4ece-976f-cc072ad4fc34}
         Original Volume: (C:)\\?\Volume{f87e43cb-3ff1-11de-9b80-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
         Originating Machine: WIN-V5MTNQS4ECM
         Service Machine: WIN-V5MTNQS4ECM
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
l, Auto recovered

Contents of shadow copy set ID: {0a4d54a6-0fb5-4104-bb2e-67db5398be42}
   Contained 1 shadow copies at creation time: 8/13/2009 8:06:44 PM
      Shadow Copy ID: {dd68f268-6ca3-44dc-a089-3af24c3a9a6f}
         Original Volume: (C:)\\?\Volume{f87e43cb-3ff1-11de-9b80-806e6f6e6963}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
         Originating Machine: WIN-V5MTNQS4ECM
         Service Machine: WIN-V5MTNQS4ECM
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differentia
l, Auto recovered
```

Creating a bodyfile from ShadowCopy

```
fls -r -m Shadow1 \\.\HarddiskVolumeShadowCopy1 >>
\\SIFTWORKSTATION\cases\ShadowTime\bodyfile
```

```
D:\winexercise\windows forensic tools\sleuthkit-windows>fls -r -m Shadow1 \\.\Ha
rddiskVolumeShadowCopy1 >> \\SIFTWORKSTATION\cases\ShadowTime\bodyfile
```

```
fls -r -m Shadow2 \\.\HarddiskVolumeShadowCopy2 >>
\\SIFTWORKSTATION\cases\ShadowTime\bodyfile
```

```
D:\winexercise\windows forensic tools\sleuthkit-windows>fls -r -m Shadow2 \\.\Ha
rddiskVolumeShadowCopy2 >> \\SIFTWORKSTATION\cases\ShadowTime\bodyfile
```


This page intentionally left blank.

Creating the Timeline

```
$ mactime -d -b bodyfile > shadow_timeline.csv
```

Wed Apr 22 2009 01:57:21	262144	...b	44760-128-3	C:/Windows/System32/config/sam
Wed Apr 22 2009 01:57:21	262144	...b	44760-128-3	Shadow1/Windows/System32/config/sam
Wed Apr 22 2009 01:57:21	262144	...b	44760-128-3	Shadow2/Windows/System32/config/sam
Wed May 13 2009 13:19:59	262144	.a..	44760-128-3	Shadow1/Windows/System32/config/sam
Wed May 13 2009 13:19:59	262144	.a..	44760-128-3	Shadow2/Windows/System32/config/sam
Thu Jul 16 2009 20:50:24	262144	m.c.	44760-128-3	Shadow1/Windows/System32/config/sam
Fri Jul 17 2009 15:01:28	262144	m.c.	44760-128-3	Shadow2/Windows/System32/config/sam
Tue Mar 16 2010 00:31:08	262144	.a..	44760-128-3	C:/Windows/System32/config/sam
Tue Mar 16 2010 00:33:42	262144	m.c.	44760-128-3	C:/Windows/System32/config/sam

Switch over to your SIFT Workstation and run mactime. I usually run it with the comma-delimited format so I can view it in excel.

```
sansforensics@SIFT-Workstation:/cases/ShadowTime$ mactime -d -b bodyfile >
shadow_timeline.csv
```

One thing I immediately noticed was that with all three times that are the same you will have three separate entries. However, you are also able to pinpoint which shadow file your data might reside in. Let us take a look at the timeline of the SAM hive alone without any other files.

Once you have created your timeline, you will notice the multiple entries that are created from a single file. In order to show the example properly, we will only focus on the SAM hive file. We have the entries from the current system, shadow1, and shadow2 showing quite nicely.

Notice that all three entries share the same file creation (birthdate) on Wed Apr 22, 2009 01:57:21. While this is useful condensing this duplicate information that shows the single creation time entry might be sought. We also see that the file has 3 separate last data modification times.

Establishing an overall timeline is easy, but you also have a lot of repetition and duplicate entries which could become distracting when you pull in the times from across the entire system and all of the shadow volumes. It is at this point I decided to create two timelines. The first is created from the above, the 2nd is created to reduce the amount of entries.

One option to consider is to compile all the data into one directory listing using C:\\ across each VolumeShadow. I re-ran the shadow timeline tools and created a single timeline without an identifier where the data might reside (e.g. Shadow1 or Shadow2). Viewing is much less of a headache via this method, but it is up to the investigator to choose which way might work best.

Wed Apr 22 2009 01:57:21	262144	...b	44760-128-3	C:/Windows/System32/config/sam
Wed Apr 22 2009 01:57:21	262144	...b	44760-128-3	Shadow1/Windows/System32/config/sam
Wed Apr 22 2009 01:57:21	262144	...b	44760-128-3	Shadow2/Windows/System32/config/sam
Wed May 13 2009 13:19:59	262144	.a..	44760-128-3	Shadow1/Windows/System32/config/sam
Wed May 13 2009 13:19:59	262144	.a..	44760-128-3	Shadow2/Windows/System32/config/sam
Thu Jul 16 2009 20:50:24	262144	m.c.	44760-128-3	Shadow1/Windows/System32/config/sam
Fri Jul 17 2009 15:01:28	262144	m.c.	44760-128-3	Shadow2/Windows/System32/config/sam
Tue Mar 16 2010 00:31:08	262144	.a..	44760-128-3	C:/Windows/System32/config/sam
Tue Mar 16 2010 00:33:42	262144	m.c.	44760-128-3	C:/Windows/System32/config/sam