508.5

IR & Hunting Across the Enterprise Advanced Adversary & Anti-Forensics Detection

SANS

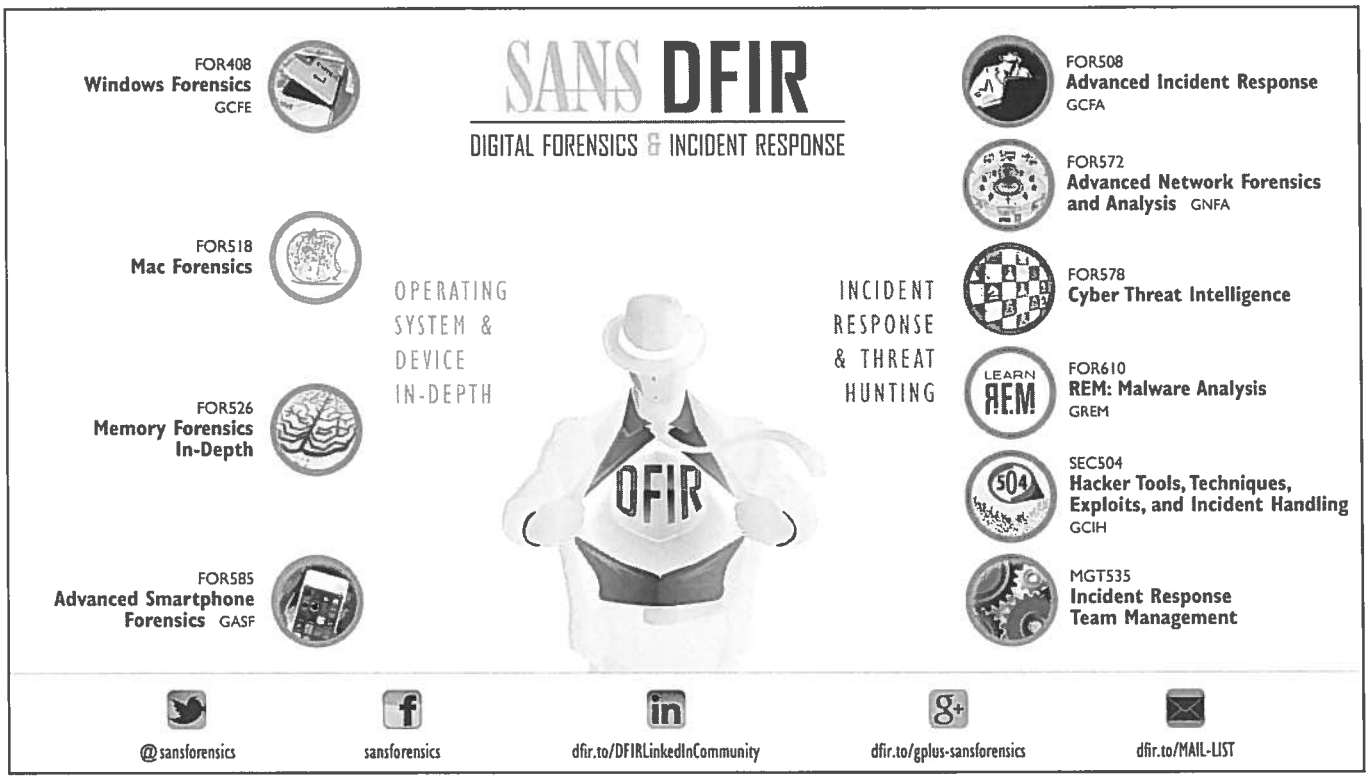
THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

FOR508.5 | Advanced Digital Forensics and Incident Response

IR & Hunting Across the Enterprise | Advanced Adversary & Anti-Forensics Detection

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

http://dfir.sans.org



This page intentionally left blank.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

OPERATING SYSTEM & DEVICE IN-DEPTH

FOR408
Windows Forensics

FOR518
Mac Forensics

FOR526
Memory Forensics In-Depth

FOR585
Advanced Smartphone Forensics

INCIDENT RESPONSE & ADVERSARY HUNTING

FOR508
Advanced Incident Response

FOR572
Advanced Network Forensics and Analysis

FOR578
Cyber Threat Intelligence

FOR610
REM: Malware Analysis

SEC504
Hacker Tools, Techniques, Exploits, and Incident Handling

MGT535
Incident Response Team Management

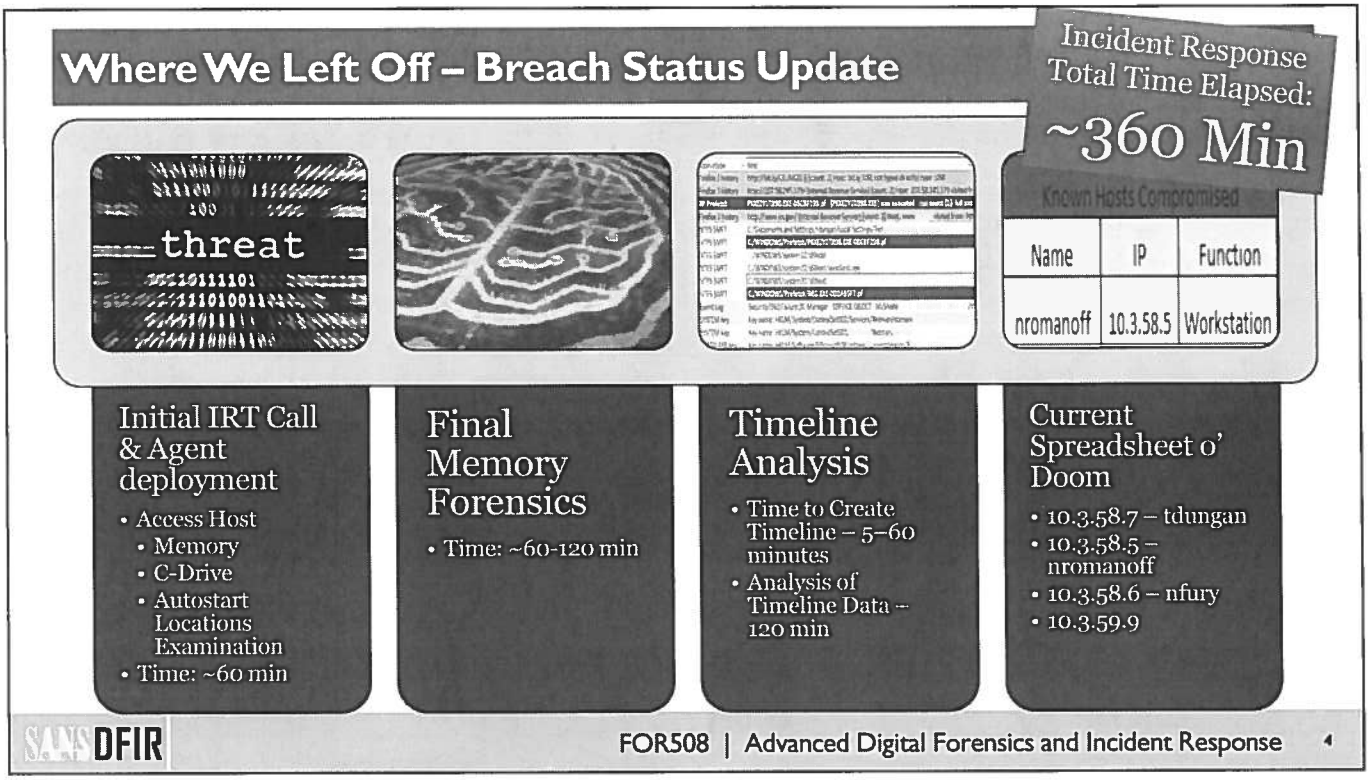
@sansforensics

sansforensics

dfir.to/DFIRLinkedInCommunity

dfir.to/gplus-sansforensics

dfir.to/MAIL-LIST



This page intentionally left blank.

Post-Timeline Analysis: Breach Status Update Host Indicators

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

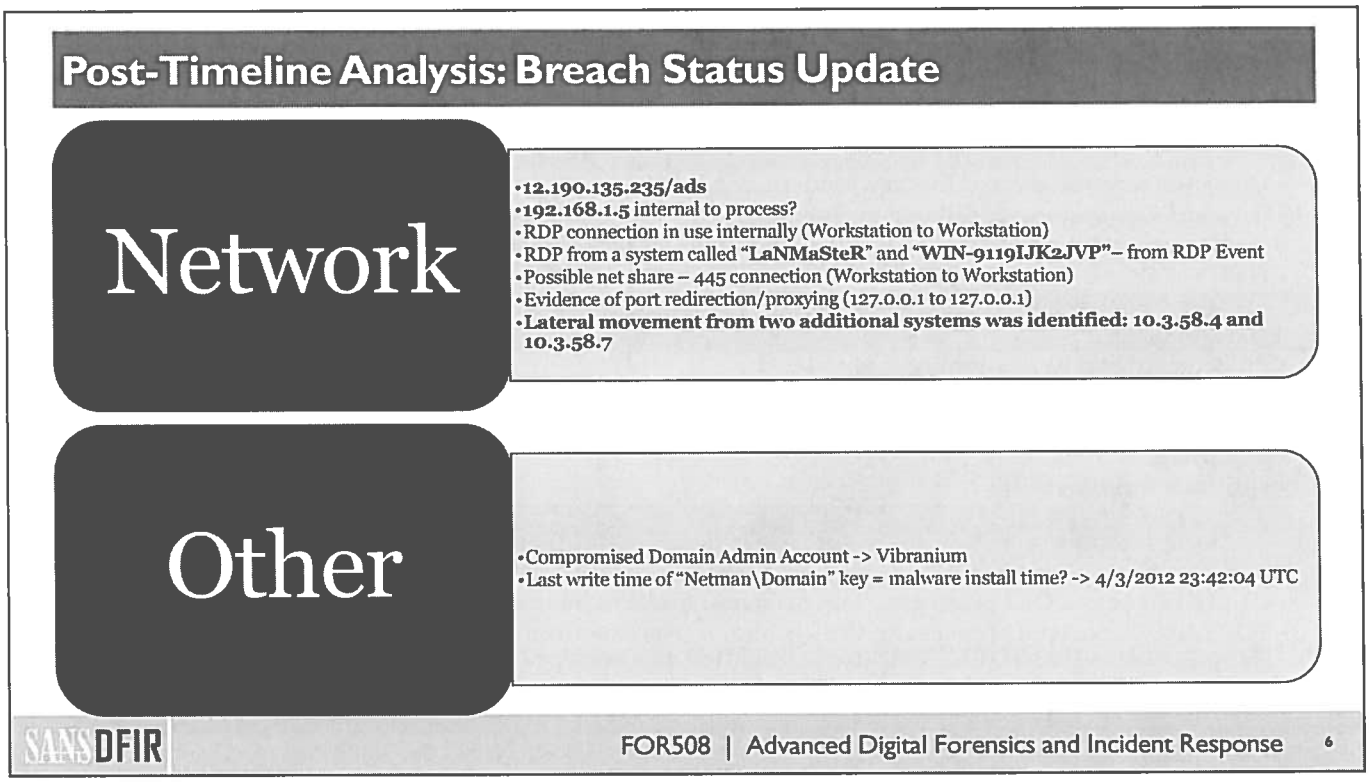
spinlock.exe

- Possible lateral movement tool of choice.
- C:\windows\system32\spinlock.exe
- python25.dll
- bz2.pyd
- spinlock.manifest

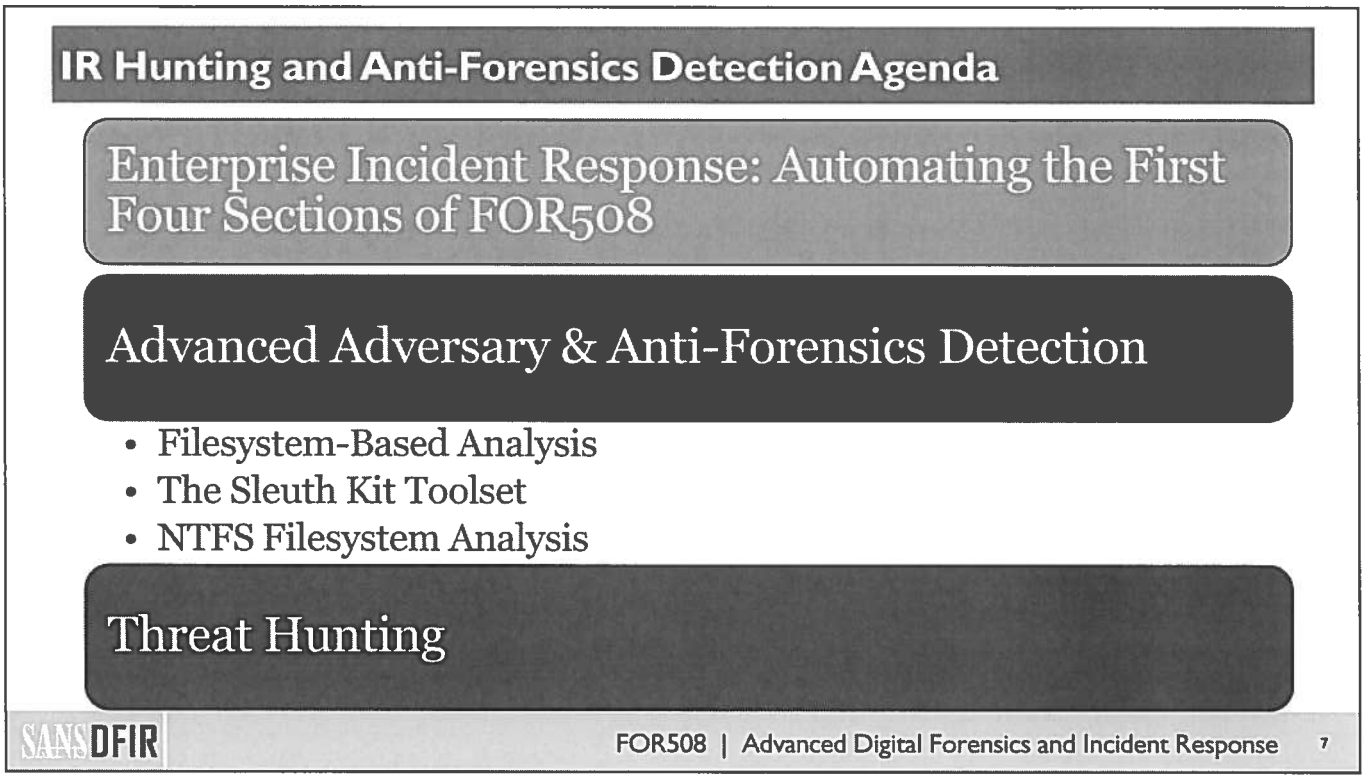
Additional Files

- DLLHOT.exe is a GUI program. The program looks to be used for AV Bypass.
- TOPLZAGU.exe is dropped onto the system, it was executed at 4/3/2016 21:03:30. The Event ID 7045 shows that TOPLZAGU.exe is installed as a service.
- PSEXECSVC was executed several times in a row starting on 4/3/2012 at 21:11:07.

This page intentionally left blank.



This page intentionally left blank.



This page intentionally left blank.

FOR508.5 | Advanced Digital Forensics and Incident Response

Incident Response & Hunting Across the Enterprise

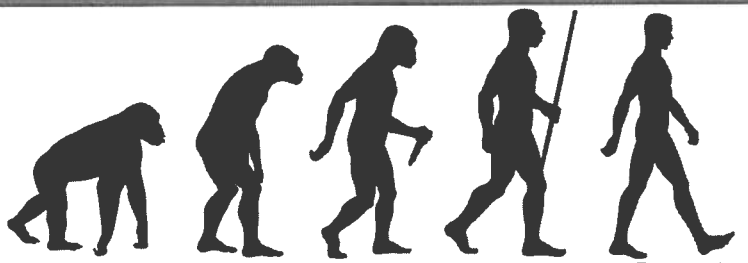
Author: Rob Lee
rlee@sans.org
http://twitter.com/robtlee
http://twitter.com/sansforensics

Enterprise Incident Response: Automating First Four Sections of FOR508

"The trouble was, innovation never resulted in victory over the long term. It was too easy for the enemy to imitate and improve on your innovations."
– Orson Scott Card, *Ender's Shadow*

This page intentionally left blank.

Evolution of IR Scripting



- Batch Files – Difficult to deploy, caches credentials, limited feature set, output hard to collect and consume
- WMI – Scales easily, safeguards credentials, incredibly granular system reporting, output in XML, HTML, CSV, and so on
- PowerShell – WMI + scripting and execution engine

If you have been doing incident response for a while, there is a good chance that you have employed batch scripts to perform live response data collection on systems. Some of you may still lovingly maintain your batch file collection, nicely organized by the Windows operating system type. If that's the case, it is time for an upgrade. Batch files have significant limitations such as being difficult to deploy at scale (without a third-party tool such as a patch manager), having a limited feature set without the inclusion of a plethora of third-party tools such as the SysInternals Suite, the inconsistent data output formats, the lack of a built-in capability to return remote data (network shares are often used), and perhaps worst of all, the fact that most implementations dangerously cache the credentials on the remote (and possibly hacked) system. In modern Windows systems, there are far better options.

Windows Management Instrumentation (WMI) has been in place and improved upon since Windows 2000. It was designed for administrative data collection of both local and remote systems. As such, it scales easily (though it will require a VB, PowerShell, or equivalent script to do so), allows detailed querying of a vast number of objects, classes, and properties throughout the OS, uses Kerberos network logons to authenticate so that credentials and tokens are not cached on the remote system, and provides a robust output capability including support for XML, HTML, and CSV formats that can be ingested into a database.[1]

Although WMI provides the mechanism for pulling data from the enterprise, PowerShell provides the scripting capability to leverage WMI. The two are inextricably linked and many of PowerShell's built-in cmdlets are simple wrappers around WMI commands. It also extends WMI's capabilities bringing in .NET capabilities, easy output post-processing, and the capability to execute arbitrary binaries to extend your collection capabilities. PowerShell and WMI are nearly the best-case scenario for Windows IR collection. In this section, we show you how to get started.

[1] "Running WMI Scripts Against Multiple Computers:" https://technet.microsoft.com/en-us/library/ee692838.aspx

Incident Response Using WMIC

- Windows Management Instrumentation CLI:
  - `/node:<remote-IP>` | `/user:<admin acct>`
- Get auto-start processes:

```
wmic /node:10.1.1.1 startup list full
```

- Remote process list:

```
wmic /node:10.1.1.1 process get
```

- Network configuration:

```
wmic /node:10.1.1.1 nicconfig get
```

- Prebuilt wmic scripts by Justin Hall on USB:
  `wmic_lr.local.cmd` & `wmic_lr.remote.cmd`

Windows Management Instrumentation Command Line Interface (WMIC) is available on all Windows systems post Win2000. It gives deep insight into a wide array of operating system information and can be easily scripted via VBScript or the newer PowerShell. WMIC allows remote queries, extending your ability to gather information across the network, so you don't have to be at the system (or at an interactive command prompt). It also provides a safe means to get a majority of our incident response data over the network. As Mike Pilkington described in his SANS Forensic Blog post, unlike PsExec, WMIC uses the native authentication mechanisms, eliminating some of the concern of exposing critical credentials.[1]

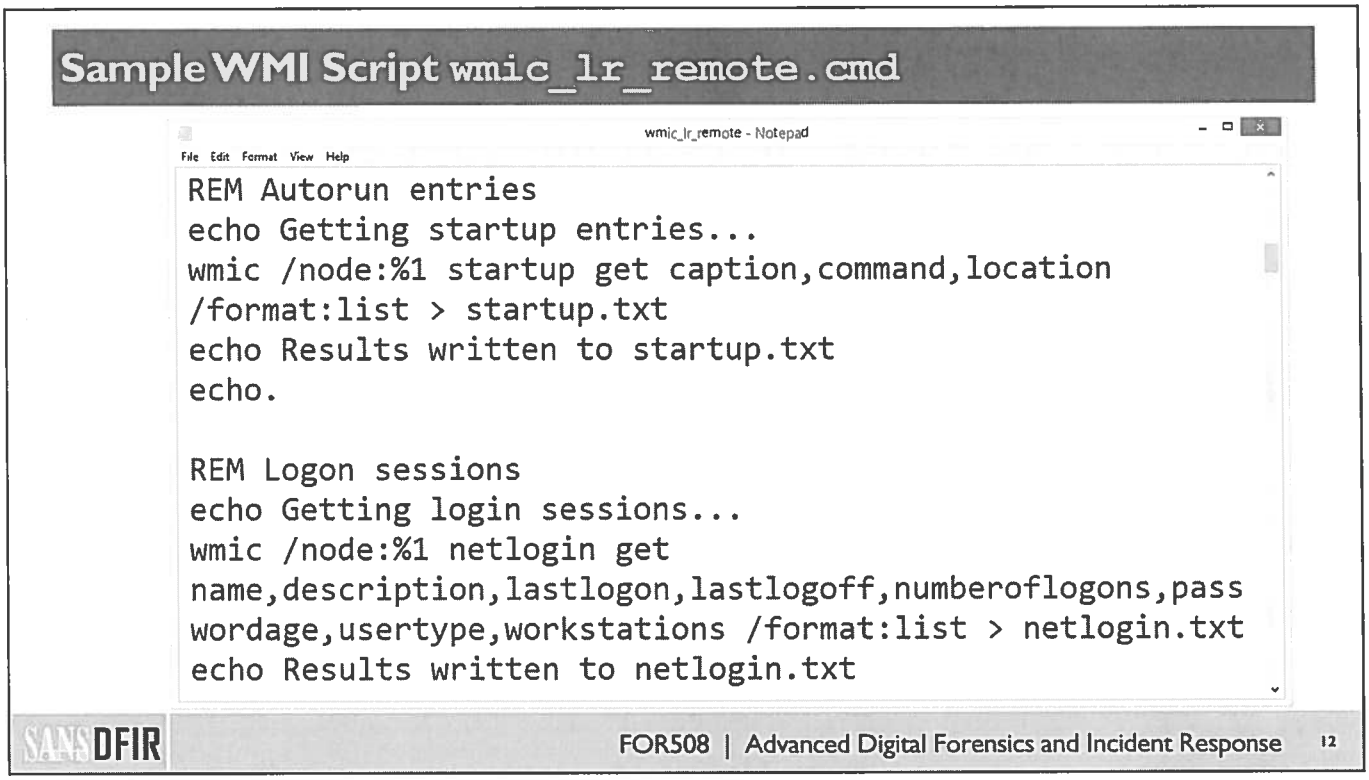
WMIC is also trivial to script, making it ideal for including in incident response scripts. For example, Justin Hall released a set of scripts via Creative Commons that provide an excellent starting point for a WMIC-based IR solution. You can find his scripts on the course USB named wmic_lr.local.cmd & wmic_lr.remote.cmd.

The vast array of options in WMIC makes getting started a bit daunting. The first command you should learn is "wmic /?." There are some terrific resources available online regarding using WMIC for incident response. A good starting point is the SANS ISC blog.[2] As an example of the power and simplicity of WMIC, the ISC blog provides the following WMIC command to spot executable running from strange locations:

```
wmic PROCESS WHERE "NOT ExecutablePath LIKE '%Windows%'" GET
ExecutablePath
```

[1] http://computer-forensics.sans.org/blog/2010/06/04/wmic-draft

[2] https://isc.sans.edu//diary.html?storyid=1622



One way to utilize WMI for data collection is to create simple scripts that run WMI commands. Justin Hall created and distributed an excellent starter set of scripts written to take advantage of WMI. Here we see his "remote" script that is designed to take a hostname or an IP address as input and run a series of WMI commands, saving the results to individual text files. The scripts are located on your USB and are a good place to start learning some of the more useful commands for IR collection. One limitation is that the remote script is designed to be run against a single remote host. Running the script via a FOR loop that iterates through an IP range or creating a script to take a file containing IP addresses to query are two ways this capability could be extended.[1]

[1] "Running WMI Scripts Against Multiple Computers": https://technet.microsoft.com/en-us/library/ee692838.aspx

The Future of IR: PowerShell

- The future of Windows administration
- Default in Win7/8/10 and Server 2008/2012/2016
- Powerful scripting language
- WMI, .NET, and COM all in one
- Allows for remote analysis and local logging
- Flexible post-processing and filtering

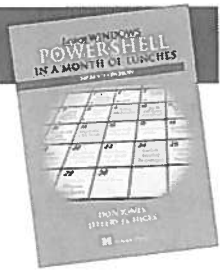
"PowerShell is both a scripting language and a powerful interactive command interface similar to Bash in UNIX. PowerShell console, where the commands are run, is similar to the Windows command interface, cmd.exe. PowerShell commands can be run in the background or interactively if a particular country's privacy policy enforces an organization to do so. PowerShell V2 is installed by default on Windows 7 operating system."

"PowerShell commands or cmdlets are based on .NET Framework objects, which means that the objects carry multiple aspects or properties of the command. These cmdlets let you access the filesystem and other Windows operating system data stores, such as the registry. PowerShell also provides access to Windows Management Instrumentation (WMI), which means that all the WMI commands that incident responders and information security professionals are familiar with can be run using PowerShell."

"Windows 7 operating system provides an option to run the PowerShell scripts remotely. Microsoft uses the industry-standard WS-Management Protocol to provide remote management features. This comes as a service in Windows 7, which can be enabled either through command line or through group policy."

Sourced from "Live Response Using PowerShell," by Sajeev Nair: http://www.sans.org/reading-room/whitepapers/forensics/live-response-powershell-34302

PowerShell Basics



- Verb-noun naming scheme
  - `Get-Process`
  - `Get-Service`
- Large number of aliases (Dir == Get-ChildItem)
  - `Get-Alias *`
- Output is objects that are passed to other cmdlets
  - `Get-Service | Out-GridView`
  - `Netstat.exe | Select-String 'Established'`
- Direct access to providers like disk and registry
  - `Get-ChildItem HKLM:Software | Format-Wide`

If you perform incident response in the Windows enterprise, the time has come to learn PowerShell (PS). Microsoft continues to double-down on support for it and a vast amount of the operating system is now being implemented as PowerShell cmdlets. Luckily, it is one of the easiest scripting languages to learn. Cmdlets (small programs or commands) are the building blocks and use a verb-noun naming scheme with the noun being an object or class of objects to do something to. If you are already familiar with the Windows command-line interface, you will be pleasantly surprised that nearly all the commands you know have been ported over to PS. This is accomplished through aliases, and there are more than 150 prebuilt aliases. (You can also add your own.) For example, "Dir" is aliased to the cmdlet "Get-ChildItem." You will also notice that native commands and executables can be run from PowerShell just as they were in cmd.exe.

One important concept in PowerShell is that command output is not in strings, as it may look in the terminal. Data is encapsulated in objects, which can then be easily passed to other cmdlets to perform additional processing. One cmdlet might generate a list of objects representing files, computers, services, and network objects; the next in the pipeline might filter and pass on just the ones that are of interest; and the next might call methods to perform actions on the objects. A simple example is output formatting. Output can be easily changed by piping one cmdlet to an outputter such as "Out-GridView," which provides a GUI interface into the data. To display the various components of an object, pipe the command into "Get-Member."

A final important concept is the idea of a provider. PowerShell abstracts collections of items into containers called providers. Providers can be file volumes, the registry, Active Directory, and so on. This makes it trivial to interact with other objects in Windows. In the example on this slide, "Get-ChildItem HKLM:Software," we list the registry keys below the HKLM/Software hive. You can even mount a network share to a PS provider (using the New-PSDrive cmdlet)!

There are an amazing number of great resources for learning PowerShell. One classic book is *Learn Windows PowerShell in a Month of Lunches* by Don Jones and Jeffery D. Hicks. There are also many great online articles. Some relevant ones follow.

PowerShell for incident response:

- "PowerShell – Forensic Oneliners:" http://www.ldap389.info/en/2013/06/17/powershell-forensic-onliners-regex-get-eventlog/
- http://blogs.technet.com/b/heyscriptingguy/archive/2012/07/28/weekend-scripter-using-powershell-to-aid-in-security-forensics.aspx

PowerShell to find evil:

- "Use PowerShell to Aid in Security Forensics," Technet blog: http://blogs.technet.com/b/heyscriptingguy/archive/2012/05/28/use-powershell-to-aid-in-security-forensics.aspx
- Offline Analysis: http://blogs.technet.com/b/heyscriptingguy/archive/2012/05/29/use-powershell-to-perform-offline-analysis-of-security-logs.aspx
- Get File Hashes: http://blogs.technet.com/b/heyscriptingguy/archive/2012/05/30/learn-the-easy-way-to-use-powershell-to-get-file-hashes.aspx
- Find Changed Files: http://blogs.technet.com/b/heyscriptingguy/archive/2012/05/31/use-powershell-to-compute-md5-hashes-and-find-changed-files.aspx

PowerShell Basics: Remoting

- WinRM Service required
  - Enabled by default on Server 2012+
- **`Enter-PSSession`**
  - Provides remote shell; Alternative to SSH
- **`Invoke-Command`**
  - Allows concurrent one-to-many command execution
- Filtering accomplished on remote host

One of the most useful features of PowerShell is that starting with version 2 (Win7 and above), it has native support for executing commands on remote systems. This feature is at the crux of why PS is so valuable for incident responders. The Windows Remote Management service is used, which is already enabled in many environments for WMI data collection and event log forwarding and is enabled by default in Server 2012. The transfer protocol is WS-Management (WSMAN), which uses SOAP, XML, and HTTP listeners to pass through packet inspection devices. Even though HTTP is employed for the inbound connection, the data transferred is encrypted and credentials are authenticated via Kerberos.

The cmdlet "Enter-PSSession <computername>" provides an interactive remote shell on the target system. However, unlike RDP or some implementations of PSExec, credentials are not cached on the remote system, providing an excellent solution for quick data gathering from a remote host.[1]

The "Invoke-Command" cmdlet is used to send remote tasks to systems without creating an interactive session.[2] The full suite of PS capabilities is available on the remote system including the capability to run scripts. A script can be passed via the "-ScriptBlock" parameter, or a script can be copied to the remote system using the "-FilePath" option. Although the ability to run scripts remotely is nice, its true power comes in the built-in capability to extend execution from one-to-many. Instead of needing to implement a loop in the script to iterate through a collection of systems, "Invoke-Command" can natively take an array of computer names as an argument. Now with one command you can run an extremely complicated script across potentially thousands of systems!

You may be thinking, "How long will it take to run a script on thousands of systems?" The answer is probably a lot less than you think! Jason Hofferle did a comparison of using a WMI command to retrieve the last 20 events from the security log of 100 computers using a loop versus using the concurrent connections allowed by "Invoke-Command." The results were astounding, with "Invoke-Command" completing the task on 100 systems in 15 seconds and the loop taking more than 6 hours! Further, he scaled the PS job to 1,000 systems and it completed in 131 seconds.

The big difference here is that "**`Invoke-Command`**" pushes processing and filtering onto the remote host. The WMI loop had to bring the event log all the way back to the host system where the filtering was accomplished. Pushing processing and filtering to the host is **exactly** what we want when doing large-scale IR collection! Finally, PS also includes the capability to manage remote executions as a background job with the "-AsJob" parameter. This is especially useful when running scripts that take a long time or on many systems. The status of remote jobs can be identified with the "Get-Job" cmdlet.

[1] "Power of PowerShell Remoting" by Mike Pilkington: http://digital-forensics.sans.org/blog/2013/09/03/the-power-of-powershell-remoting

[2] "Invoke-Command": https://technet.microsoft.com/en-us/library/hh849719.aspx

[3] "PowerShell Remoting Performance": http://www.hofferle.com/powershell-remoting-performance/

PowerShell Basics: Authentication

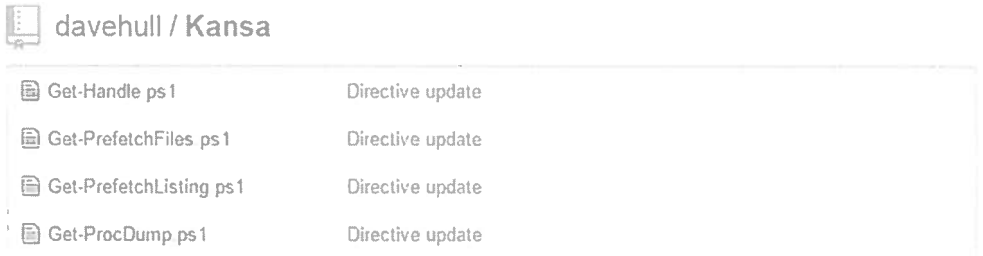
- Best-case scenario from security perspective
- Default is nondelegated Kerberos
  - Precludes delegate token stealing
  - Do NOT use CredSSP (dual-hop) auth
- Non-interactive (Type 3) logon
  - Even `Enter-PSSession` does not cache creds
- Credentials not passed to remote system and hence not available to tools, such as mimikatz, incognito, etc.

Strong authentication and encryption are taken care of in the background by PowerShell. Remote PS sessions are encrypted and authentication occurs using Kerberos. Sessions are treated as non-interactive (including Enter-PSSession shell), so password hashes are not cached on the remote system. This makes it far safer to use than something such as RDP. Also by default, PS uses nondelegated authentication tokens, which means that tokens cannot be stolen from the remote system and then used to authenticate to additional systems. This is the ideal situation from a security perspective but can cause issues if scripts are written in such a way that they require access to a third system. This is called "dual-hop" authentication and is implemented with a service called CredSSP (essentially single sign-on). However, you should aim to NEVER use it. CredSSP will store credentials on the remote system, including the hash and plaintext password extracted by tools such as mimikatz.[1] In fact, this is one of the reasons why PS is such a great replacement for standard batch scripts. When run on remote hosts, batch scripts often required the host to authenticate to a network share to either pull binaries or store data, effectively requiring that "dual-hop" authentication be used. In general, scripts that pull binaries from shares or dump data to shares require delegation authority, which leaves credentials available on the remote host! By default PowerShell remoting eliminates these security concerns and ensures that valuable credentials such as Domain Admin are not scattered on systems throughout the environment.

[1] http://www.powershellmagazine.com/2014/03/06/accidental-sabotage-beware-of-credssp/

Kansa: PowerShell IR Framework

- Designed by Dave Hull to support massive IR data collections
- Framework organizes data collection and module selection
  - Modules are written as PowerShell scripts
- Easily scales to thousands of systems via PowerShell remoting
- Not confined to PowerShell cmdlets – exec virtually anything



Kansa uses PowerShell Remoting to run user contributed modules across hosts in an enterprise to collect data for use during incident response, breach hunts, or for building an environmental baseline.

- If users don't supply their own list of remote systems (targets), Kansa will query Domain Controllers and build the list automatically.
- Error handling and transcription with errors are written to a central file.
- PowerShell remote job management is used, invoking each module on target systems, in parallel, currently using PowerShell's default of 32 systems at a time.
- Output management —As the data comes in from each target, Kansa writes the output to a user-specified folder, one file per target per module.
- A modules.conf file where users can specify which modules they want to run and in what order, lending support to the principal of collecting data in the order of volatility.

Kansa was designed to gather data from thousands of hosts at a time, given two prerequisites are satisfied: 1) your targets are configured for Windows Remoting (WinRM) and 2) the account you're using has Admin access to the remote hosts.

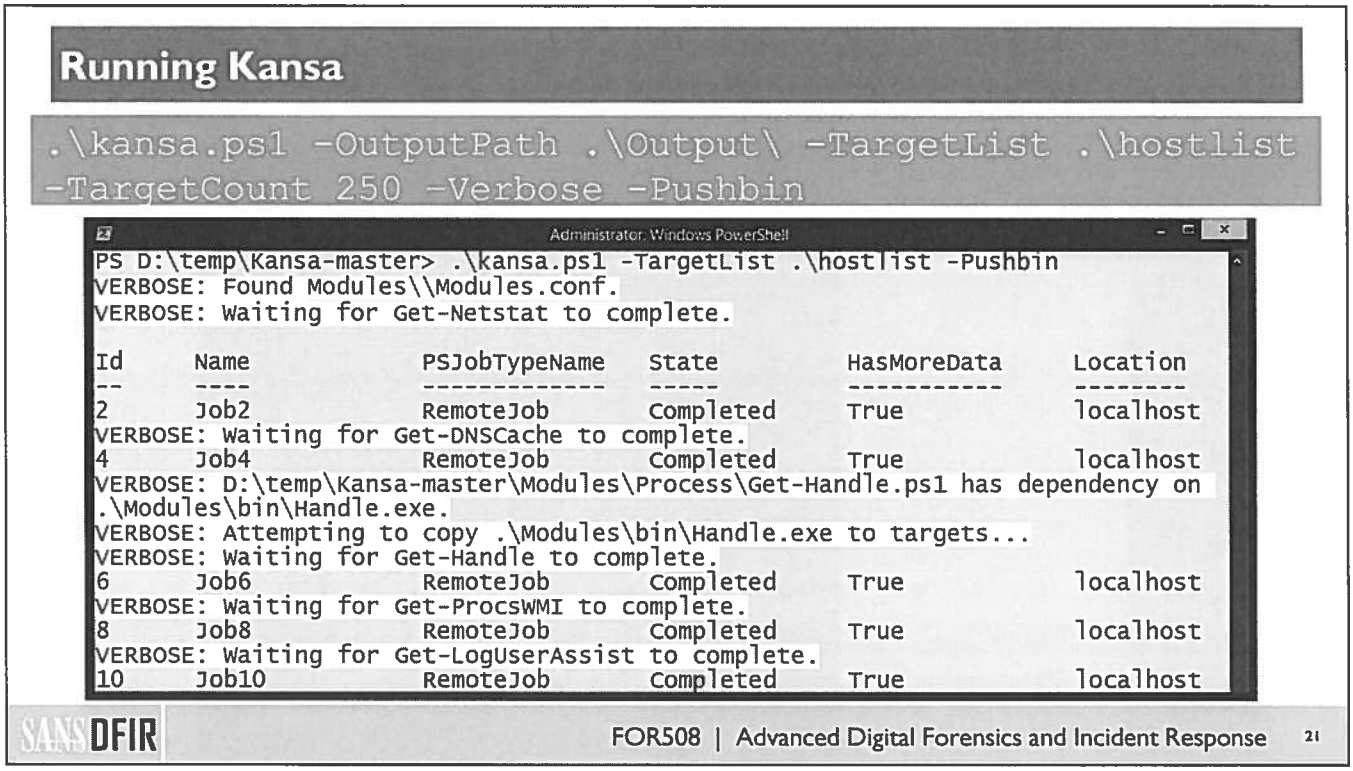
Getting a copy of Kansa is easy; <https://github.com/davehull/Kansa/archive/master.zip> will pull down a zipped copy of the master repository. Simply download it and extract it to a location of your choice. The repository contains several folders.

The .\Modules folder contains the plugins that Kansa will invoke on remote hosts. Plugins are aggregated into subfolders by the type of data they are designed to extract. Looking around in this folder will give you a good idea of the current capabilities of the tool.

The `.\Analysis` folder contains PowerShell scripts for conducting basic analysis of the collected data. Many of the analysis scripts require **`logparser.exe`**, which is not part of the distro. So make sure you also download and place logparser.exe in your path. Keep in mind that you can also create your own analysis workflow. Because Kansa output is typically tab-separated, it can easily be imported into the database of your choice; you can also run queries against it using Logparser, load it into Excel, and so on.

Kansa's capabilities can be easily expanded by writing new modules. Because the language is PowerShell, the barrier to entry is low and the capabilities available are massive. Should you modify or write new plugins, consider submitting them back to the project. Dave has put a lot of effort into this framework and would love to hear how it is being used and see more user submitted scripts!

Some text used with permission from Dave Hull is at http://trustedsignal.blogspot.com/ and https://github.com/davehull/Kansa.



A simple command line for Kansa might look like the following:

```
PS C:\tools\Kansa> .\kansa.ps1 -TargetList .\hostlist -Pushbin
```

Although the .\Modules folder holds all the available scripts to run, **`kansa.ps1`** is the script used for kicking off data collection. The command-line arguments are straightforward. "-TargetList .\hostlist" tells **`kansa.ps1`** to run collectors against the systems listed in the hostlist file, which should contain one host per line. If you omit this argument, Kansa will query **Active Directory** (AD) for the list of computers and target all of them. Querying AD in this way requires this **Active Directory** module that's bundled with Remote Server Administration Tools, so you'll need that installed if you're not providing a list of targets via -TargetList.[1] If using the AD option, you should also consider including the "-TargetCount" parameter to limit the total number of systems queried (perhaps using a small sample for a test run, and subsequently omitting it to run across the entire environment).

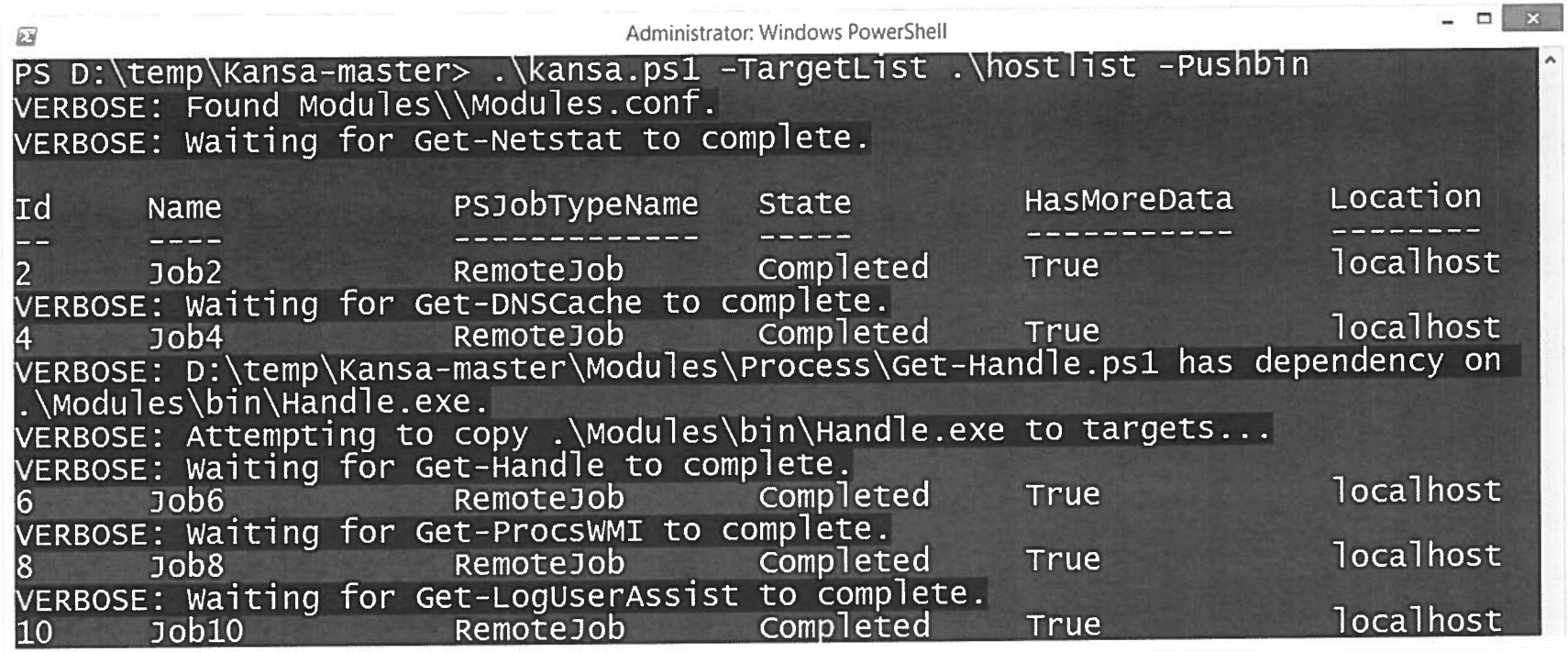
The results returned by each system will be written to an Output folder within the Kansa folder by default. You can specify a different location using the "-OutputPath" option. Each collector's output is written to its own folder. Collector scripts are typically written to take the PowerShell objects returned from remote systems and convert them to easy to parse TSV.

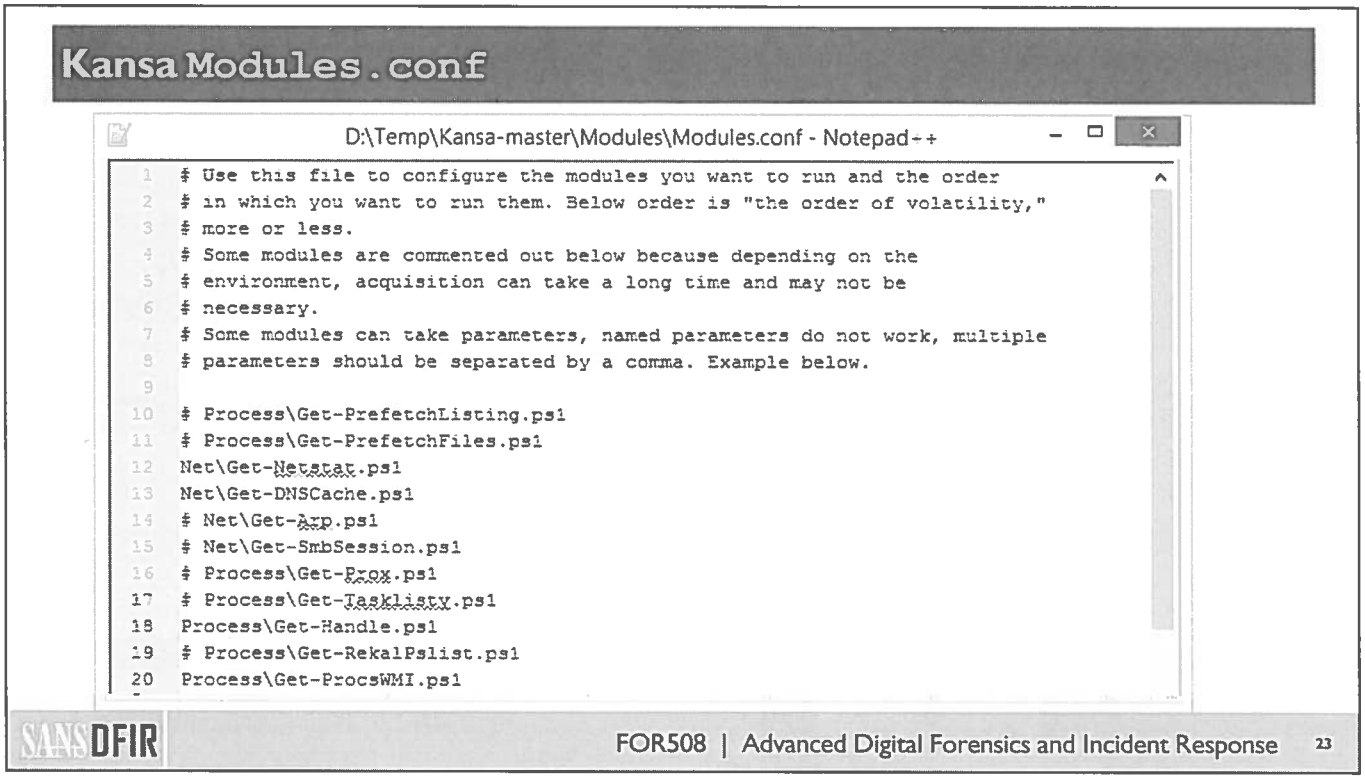
A final important option is "-Pushbin," which is required by scripts that employ third-party binaries. If this argument is provided and a script with a binary dependency is run, **`kansa.ps1`** will first copy any required third-party binaries to the targets before running the script. All in all, this is a handy way to run third-party tools remotely!

The "Verbose" option provides additional debugging information to the terminal.

[1] http://www.microsoft.com/en-us/download/details.aspx?id=39296

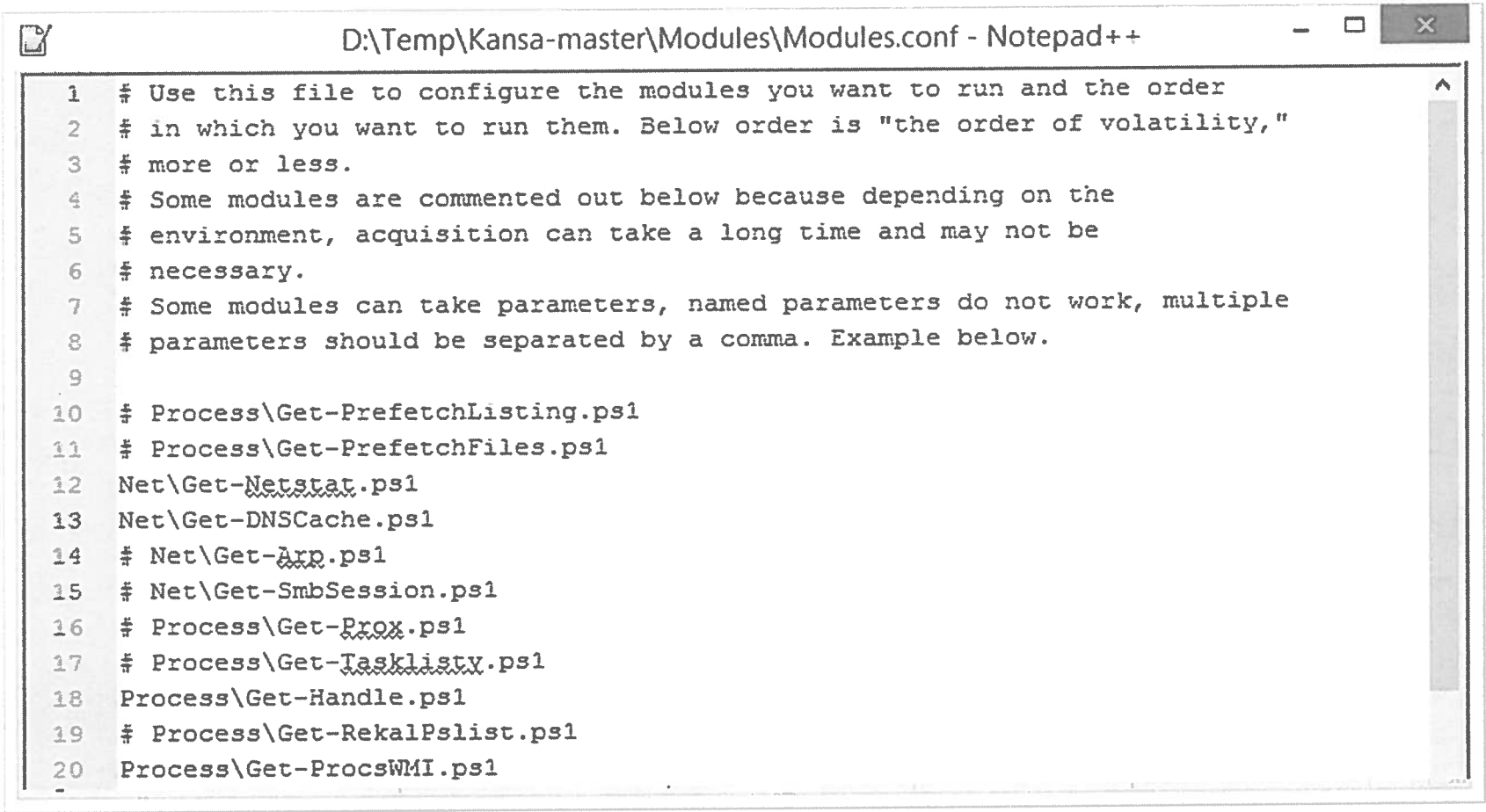
```
.\kansa.ps1 -OutputPath .\Output\ -TargetList
.\hostlist -TargetCount 250 -Verbose -Pushbin
```





All Kansa modules are written in PowerShell and are selected at runtime by referencing the Modules.conf file within the Modules folder. This file is a simple flat configuration file that contains the name and relative location (notice that subfolder names are included with the script names) of each script to be run. As new scripts are written or added to the repository, Modules.conf can be easily updated to request those modules be run. In addition, to prevent a module from running, preface the line with a "#," effectively commenting it out of the file.

The Modules.conf file should be set up to respect the Order of Volatility of data collected. Artifacts that change frequently such as network data, system memory, or Prefetch should be collected early in the process so that evidence is not unnecessarily overwritten. Simply moving those script names closer to the top of the file ensures they are run before others. For example, note that in this slide, we see scripts listed at the top, which are responsible for collecting highly volatile data such as Prefetch, DNScache, and Arp information.


D:\Temp\Kansa-master\Modules\Modules.conf - Notepad++
1 # Use this file to configure the modules you want to run and the order
2 # in which you want to run them. Below order is "the order of volatility,"
3 # more or less.
4 # Some modules are commented out below because depending on the
5 # environment, acquisition can take a long time and may not be
6 # necessary.
7 # Some modules can take parameters, named parameters do not work, multiple
8 # parameters should be separated by a comma. Example below.
9
10 # Process\Get-PrefetchListing.ps1
11 # Process\Get-PrefetchFiles.ps1
12 Net\Get-Netstat.ps1
13 Net\Get-DNSCache.ps1
14 # Net\Get-Arp.ps1
15 # Net\Get-SmbSession.ps1
16 # Process\Get-Prox.ps1
17 # Process\Get-Tasklisty.ps1
18 Process\Get-Handle.ps1
19 # Process\Get-RekalPslist.ps1
20 Process\Get-ProcsWMI.ps1

Kansa + Third-Party Tools

- Kansa is not limited to WMI or PS cmdlets
- It can also push and execute binaries
  - Place in the `.\Modules\bin` directory
  - Use `-Pushbin` argument
  - Remove binaries after execution with `-Rmbin`
- Output returned and formatted as PS objects
- Binaries can be deleted or left for future use

Get-Autorunsc.ps1	**Get-FlsBodyfile.ps1**	**Get-ProcDump.ps1**
Get-CertStore.ps1	**Get-Handle.ps1**	**Get-RekalPslist.ps1**

Although PowerShell is extremely powerful, there are certainly instances in which a third-party collection tool is necessary. Luckily, Kansa is not limited to Windows native commands or PowerShell cmdlets. The process to include a third-party binary in a script is simple: Copy the binary into the .\Modules\bin\ directory, include a special comment on the second line of the collector script, and ensure you are running kansa.ps1 with the –Pushbin argument.

The special comment that must be included in the script is referred by the author as a directive. The directive for the `Get-Autorunsc.ps1` module looks like this:

```
# BINDEP .\Modules\bin\Autorunsc.exe
```

Note that it is an actual comment, but that Kansa knows to look for the "BINDEP," which stands for "binary dependency." This directive tells Kansa to copy .\Modules\bin\Autorunsc.exe to the remote targets before executing the script. By default, binaries will be copied to the %SystemRoot% folder (usually C:\Windows). It is up to the script author as to whether the binary should be deleted after use. One reason that deletion may not be desired is if you plan to run the script again in the future and want to speed up the process. If the binary is already in place on the target system, you can omit the –Pushbin argument. If you would like to remove the binary after use, add the –Rmbin argument.

Several modules currently depend on third-party binaries, including:

- Get-Autorunsc.ps1
- Get-CertStore.ps1
- Get-FlsBodyfile.ps1
- Get-Handle.ps1
- Get-ProcDump.ps1
- Get-RekalPslist.ps1

Kansa Analysis Scripts

```
if (Get-Command logparser.exe) {
    $lpquery = @"
    SELECT
        COUNT(ImagePath, LaunchString, MD5) as ct,
        ImagePath,
        LaunchString,
        MD5,
        Publisher
    FROM
        *autorunsc.tsv
    WHERE
        Publisher not like '(Verified)%' and
        (ImagePath not like 'File not found%')
    GROUP BY
        Imag
        Laur
        MD5,
        Publ
    ORDER BY
        ct A
```

LogParser command to stack unsigned Autoruns output from multiple Kansa output files

Results from 10 domain controllers shown below (note: no outliers)

ct	Image Path	MD5
10	c:\windows\system32\cpqnimgt\cpqnimgt.exe	78af816051e512844aa98f23fa9e9ab5
10	c:\hp\hpsmh\data\cgi-bin\vcagent\vcagent.exe	54879ccbd9bd262f20b58f79cf539b3f
10	c:\windows\system32\cpqmgmt\cqmgstor\cqmgstor.exe	60668a25cfa2f1882bee8cf2ecc1b897
10	c:\program files\hpwbem\storage\service\hpwmistor.exe	202274cb14edaee27862c6ebce3128d8
10	c:\hp\hpsmh\bin\smhstart.exe	5c74c7c4dc9f78255cae78cd9bf7da63
10	c:\msnipak\win2012sp0\asr\configureasr.vbs	197a28adb0b404fed01e9b67568a8b5e
10	c:\program files\hp\cissesrv\cissesrv.exe	bf68a382c43a5721eef03ff45faece4a

Although Kansa is a data collection framework, Dave Hull had the foresight to plan for it to also facilitate analysis. Imagine that you just did a Kansa run of a 1,000 systems. How would you begin to analyze that data? A database or tool such as Splunk comes to mind, but because the data is in simple text format, PowerShell could also be used for parsing. Several sample analysis scripts are in the .\Analysis folder of Kansa. We call them samples because they are just a small collection of what could be hundreds of scripts, each designed to find one type of anomaly or signature.

Many of the current scripts rely upon the free Microsoft tool Log Parser. These scripts expect logparser.exe to be in the system path so that it can find it.

On this slide, we see part of the **`Get-ASEPImagePathLaunchStringMD5UnsignedStack.ps1`** script. The SQL-like language you see with "SELECT" and similar commands is the SQL input that Log Parser expects. Log Parser is a powerful tool and can be learned relatively quickly.[1] In this example, it was used to perform frequency counting of unsigned binaries from the Autoruns output provided by running **`.\Modules\ASEP\Get-autorunsc.ps1`** across ten domain controllers. An analyst would be looking for outliers, such as a binary that was referenced on only one or two of the domain controllers. In this case, all ten of the domain controllers had the exact same set of binaries. (Each has a value of 10 in the "ct" column.) If any were suspicious, the next step would be to cross-reference the provided MD5 hashes with a known-goods database.

[1] "Getting Started with LogParser": http://digital-forensics.sans.org/blog/2011/02/10/computer-forensics-howto-microsoft-log-parser

```
if (Get-Command logparser.exe) {
    $lpquery = @"
    SELECT
        COUNT(ImagePath, LaunchString, MD5) as ct,
        ImagePath,
        LaunchString,
        MD5,
        Publisher
    FROM
        *autorunsc.tsv
    WHERE
        Publisher not like '(Verified)%' and
        (ImagePath not like 'File not found%')
```

LogParser command to stack unsigned Autoruns output from multiple kansa output files

Results from 10 domain controllers shown below (note: no outliers)

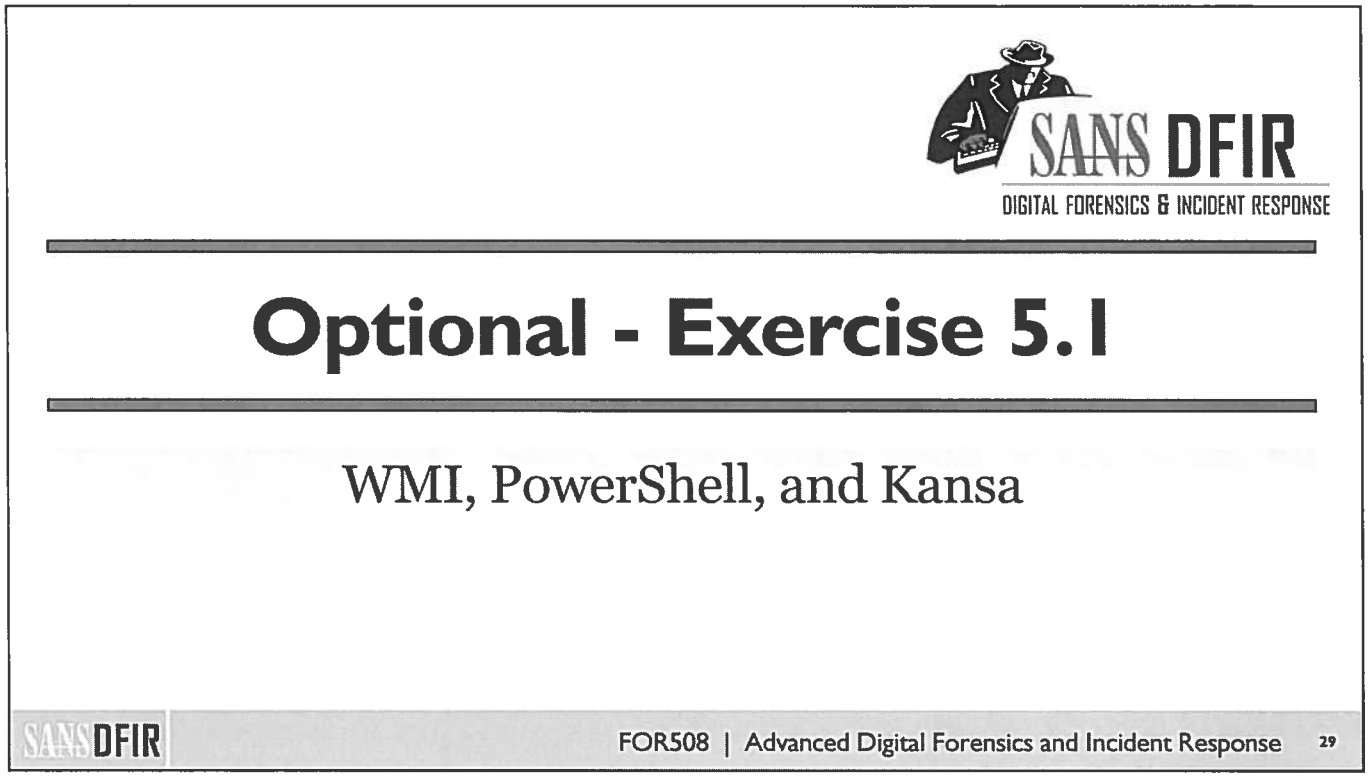
ct	Image Path	MD5
10	c:\windows\system32\cpqnimgt\cpqnimgt.exe	78af816051e512844aa98f23fa9e9ab5
10	c:\hp\hpsmh\data\cgi-bin\vcagent\vcagent.exe	54879ccbd9bd262f20b58f79cf539b3f
10	c:\windows\system32\cpqmgmt\cqmgstor\cqmgstor.exe	60668a25cfa2f1882bee8cf2ecc1b897
10	c:\program files\hpwbem\storage\service\hpwmistor.exe	202274cb14edaee27862c6ebce3128d8
10	c:\hp\hpsmh\bin\smhstart.exe	5c74c7c4dc9f78255cae78cd9bf7da63
10	c:\msnipak\win2012sp0\asr\configureasr.vbs	197a28adb0b404fed01e9b67568a8b5e
10	c:\program files\hp\cissesrv\cissesrv.exe	bf68a382c43a5721eef03ff45faece4a

What About Rootkits?

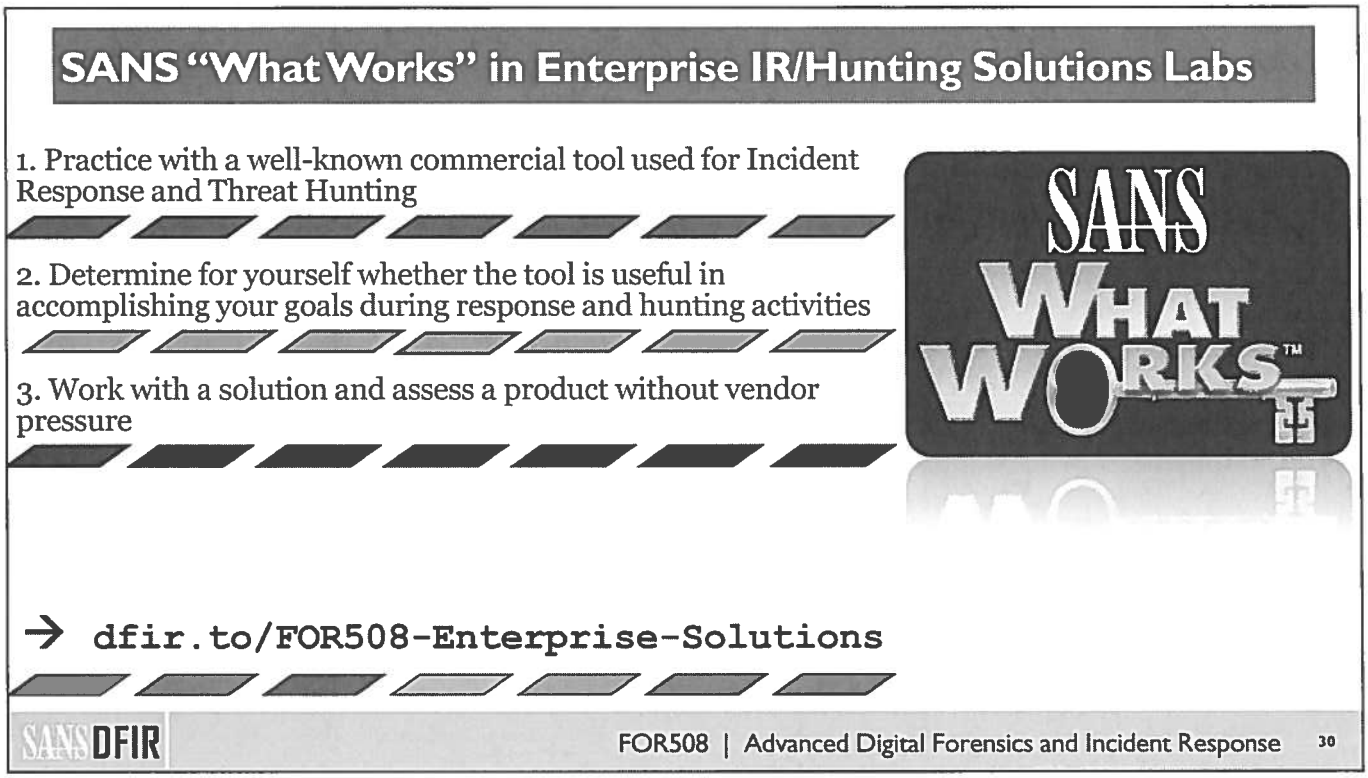
- Rootkits can defeat live response collection
  - PS and WMI rely upon Windows API
- Threat intel can help identify possibility
- Trade-off: Speed and efficiency versus completeness
- Memory analysis is your best weapon
  - Get-RekalPslist.ps1 brings mem analysis to Kansa

A common criticism of all live response collection is its susceptibility to being fooled by a malicious rootkit on the system. Rootkits are used by malware to hide its existence and often exist at low levels in the system like the kernel. Because most live response tools (including system commands, WMI, .NET, and PowerShell) rely upon the Windows API to collect data, a rootkit can easily subvert those API functions to return incomplete data. Keep in mind that rootkits are rare and are not used nearly as prevalently as some would suggest. If a rootkit is present, then data collection should be accomplished via memory or disk forensics—both of which can identify a rootkit and its activities. However, deep-dive forensics is much too time-consuming to run on hundreds of systems. Hence, we typically make the trade-off of speed and efficiency of live response toolkits for large-scale collection while augmenting that analysis with deep-dive forensics of a smaller sample of systems. Of course, knowing whether a rootkit is in play and where it might be is a chicken-and-egg problem. That question can often be answered by security tools (like the AV warning shown here) or via threat intelligence. (That is, have you been targeted by a threat group in the past with a history and capability of deploying rootkits?)

Kansa contains a proof-of-concept plugin named `Get-RekalPslist.ps1`. This module shows the best of both worlds as it is a Kansa script that can remotely run live memory forensics across multiple target systems. It accomplishes this by using the Rekall memory analysis suite with the winpmem driver that gives raw access to memory. This allows data to be collected directly from memory instead of using the Windows API, making it much more difficult for a rootkit to affect the output. It is a neat idea, but keep in mind that it pushes a 17MB zip archive to each system it is run against. This type of module is probably best used for smaller scale collection efforts against systems that are reasonably expected to have more advanced malware. One alternative is to get a complete memory image, allowing detailed analysis of the target system. We will cover those techniques in an upcoming section.



This page intentionally left blank.

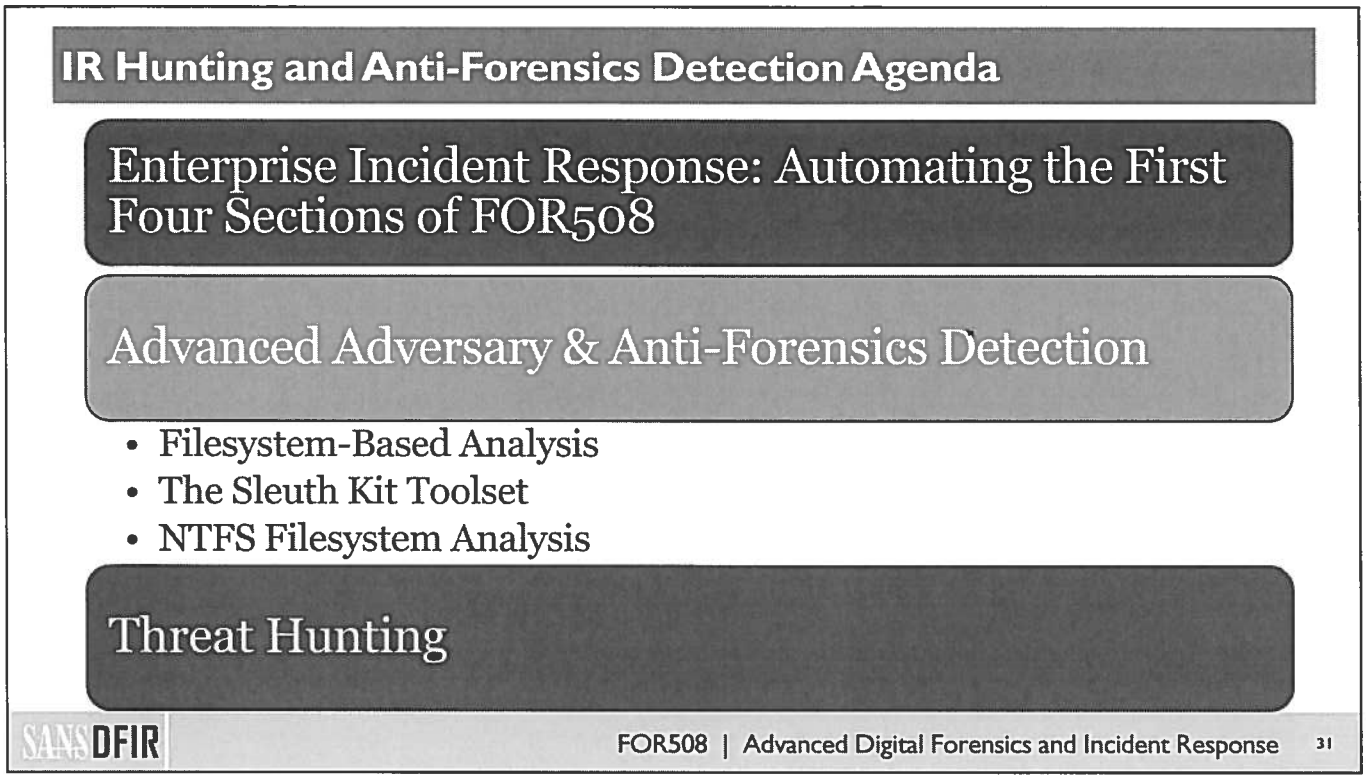


The **SANS What Works** program highlights commercial tools recommended by students as solving specific problems particularly well. This program is accomplished without any vendor sponsorship or payment for the opportunity. The program is simply meant to further the education of students and SANS attendees and expose them to real-world products to test on their own time and without influence.

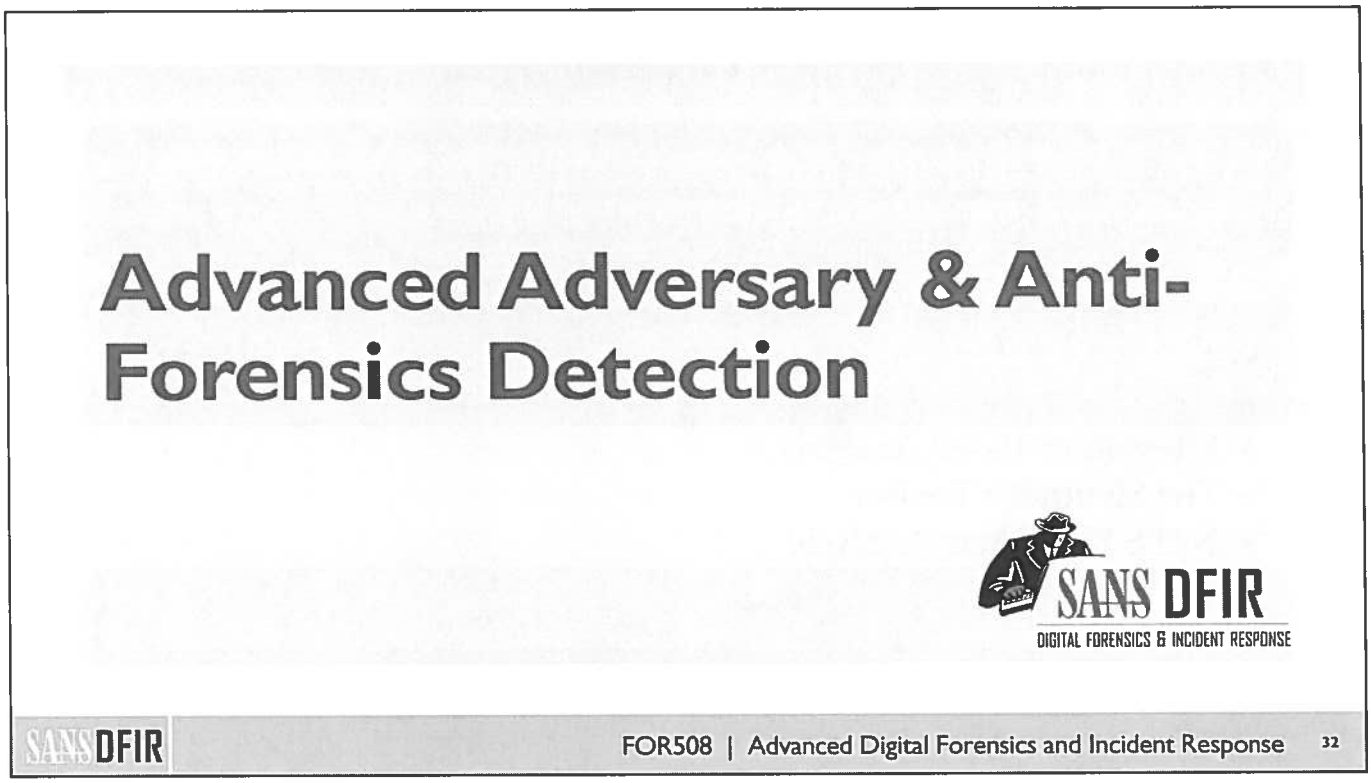
SANS DFIR has developed partnerships for many years to offer trial versions of commercial utilities so students can continue to learn and test popular solutions in their own environments. We first surveyed students to find out which solutions they feel "work" the best to solve specific cyber security challenges. We subsequently approached several vendors and challenged them to come up with training scenarios we could offer to SANS students to anonymously test the capabilities of their commercial solutions. For advanced forensics and incident response, multiple tools were recommended to help with Enterprise Incident Response and Threat Hunting. We offer this exercise not as an endorsed tool by SANS or its instructors, but as a way for each student to try the solution and ultimately assess its capabilities, good or bad, on their own. We anticipate adding additional vendor challenges in the near future.

No student information is passed to any vendor unless you specifically elect for them to know who you are. This is your chance to try a popular commercial solution without the vendor trying to sell you anything. While the lab was written by the vendor, we attempted to emphasize the same qualities that you experienced in other SANS labs. Specifically, we required that the lab should not be a "sales" presentation and should demonstrate how the tool can be used to solve real-world problems. We hope you enjoy this opportunity.

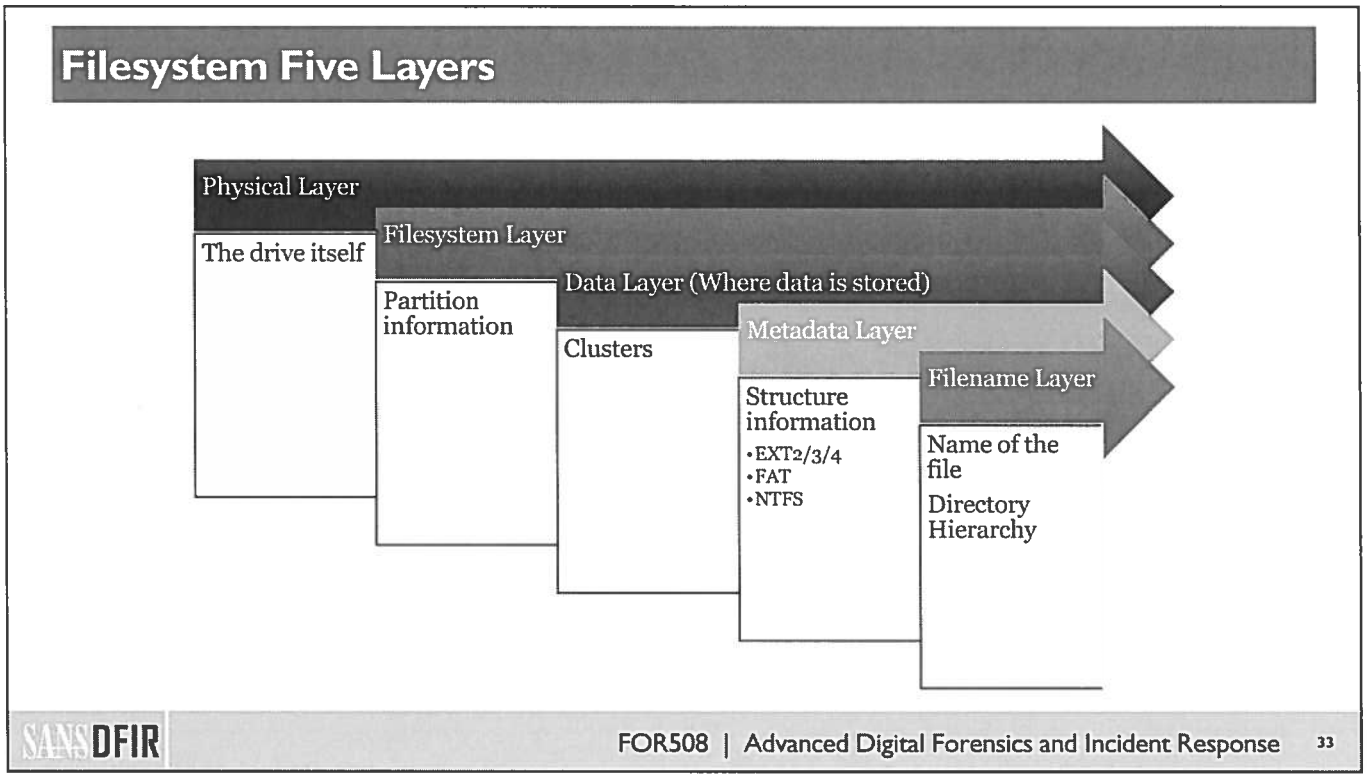
As this is not a SANS written lab, any questions regarding this lab should be directed to the provider. However, we can pass along any feedback back to them anonymously as well. Feel free to write and relate your experiences to rlee@sans.org.



This page intentionally left blank.



This page intentionally left blank.

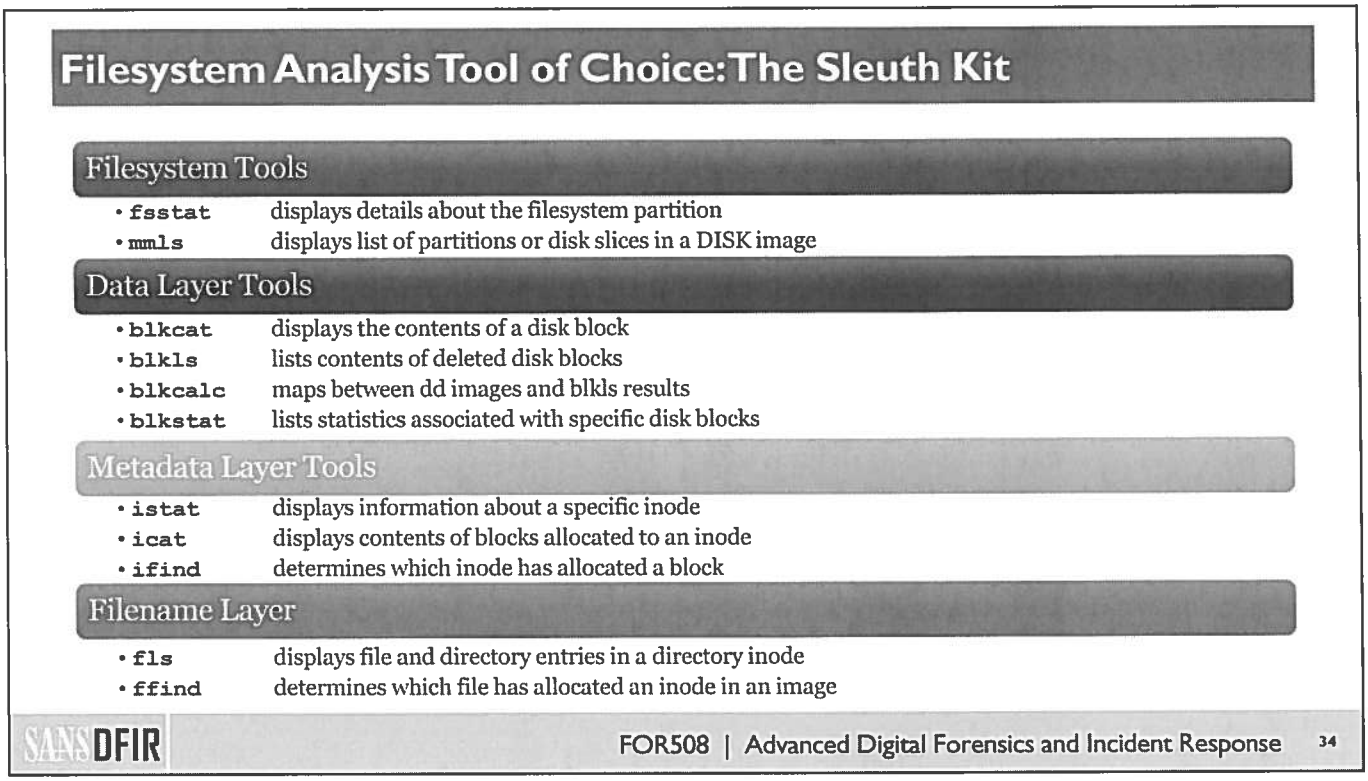


This is the overview of the five Filesystem layers that we examine over the next few sections. We start with the physical and the partition details of a drive, which lead us to the actual filesystem. The filesystem will be made up of three layers (Data, Metadata, and Filename). To understand how to best 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 filesystem forensics.

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

From the first two layers, we will examine the Physical layer; the Filesystem 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 at a high level how the three layers of the filesystem work together to actually store file data.



The Sleuth Kit 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 Filesystem layers, and the first letter of each tool name corresponds to the layer.
- The remainder of the names 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 filesystem 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 filesystem 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 more than 20 different tools sounds daunting, but the filesystem tools are organized by Filesystem and Data layers. There are a total of five 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 Filesystem 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 Filesystem 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 layers 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 filesystems 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 whether there is hidden data on a disk and whether to remove it.

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

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

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

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

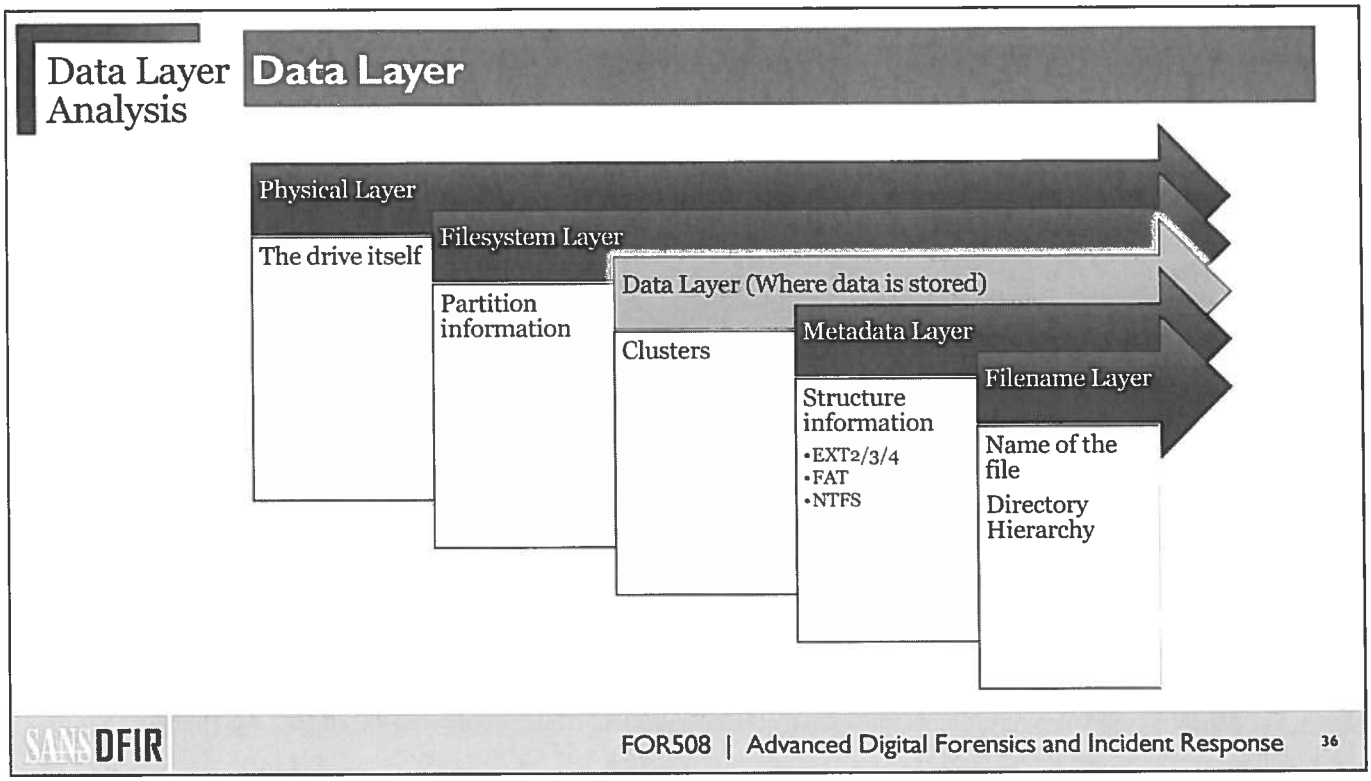
[Useful Options]

-f fstype:	Image filesystem 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 filesystem in the image (in sectors)

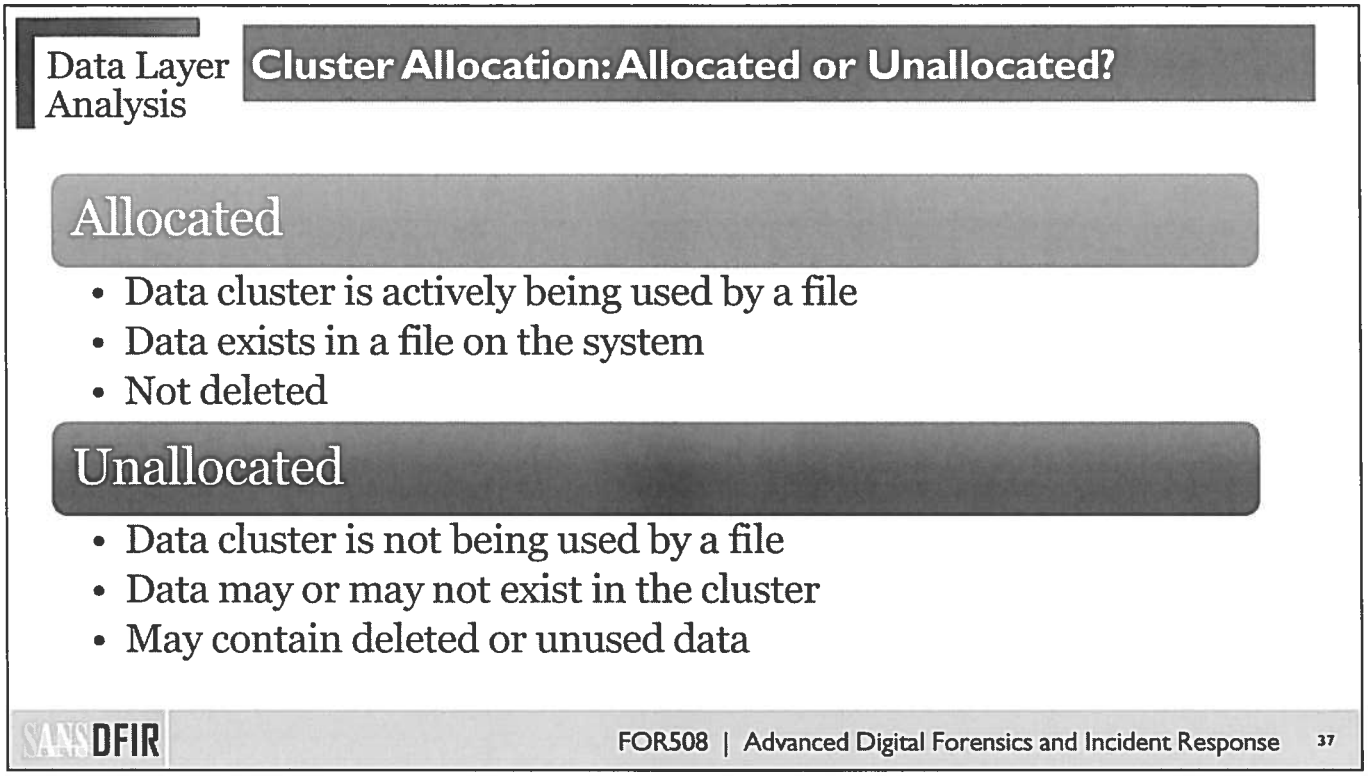
Typically, one also needs to specify the address of "something" (a data unit, metadata structure, and so on).

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 filesystem 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 wildcard can be used.

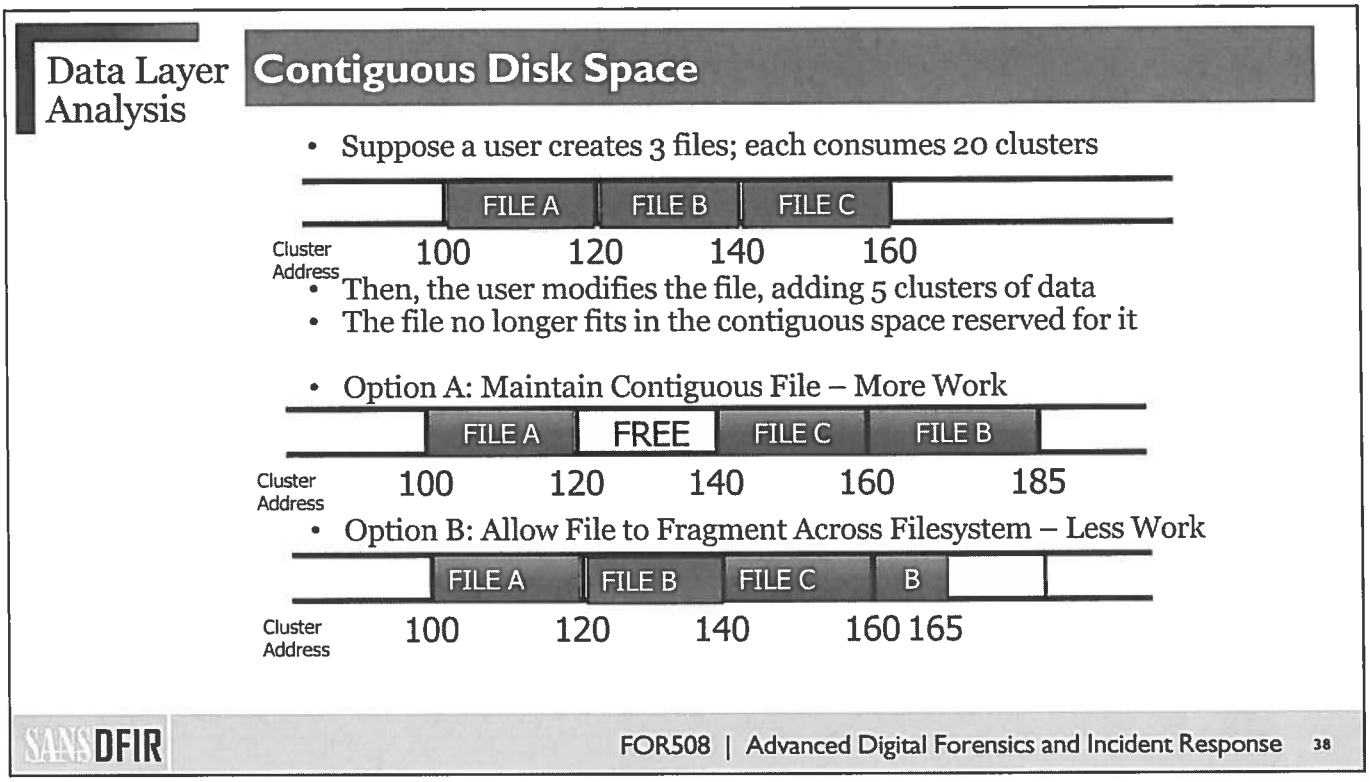


This page intentionally left blank.



Data chunks will be in one of two states on the filesystem: 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 filesystem. Even though the space is unallocated, critical evidence can be recovered from these clusters despite not being recoverable by ordinary file recovery.

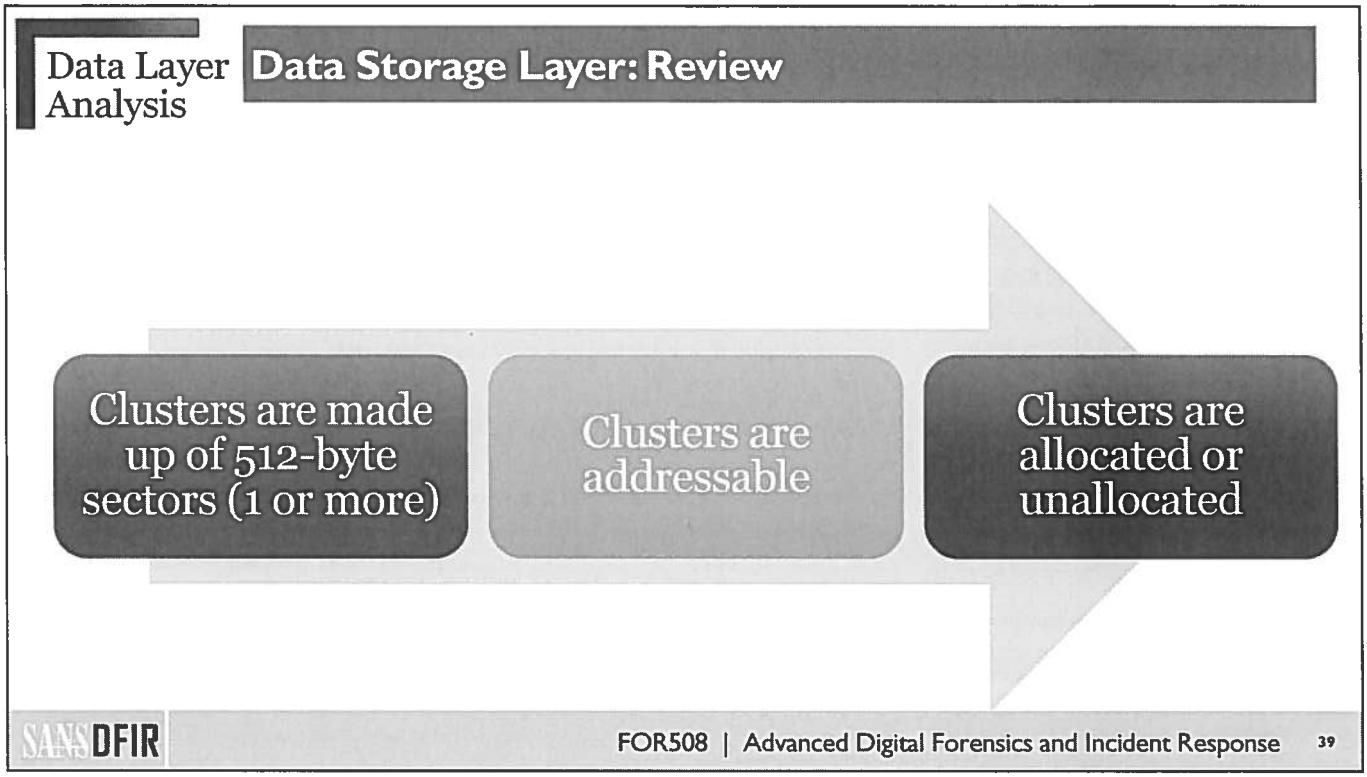
If a file is not fully recoverable, a piece of that file may still be recoverable. That piece of the file is called a file fragment. A fragment may be one or more clusters of data, but alone would not be the full file. For example, an e-mail found on a system may be recoverable; however, you may obtain only one-half of the e-mail. The other half was written over when the filesystem 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 5,000 bytes, then the disk is searched for adjacent disk clusters that comprise 5,000 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 three 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 filesystems 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 filesystem 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 due to the fact that defragmentation will rejoin fragmented files together over time.



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

Data Layer Analysis

Sleuthkit Data Layer Tools

`fsstat:`	Provides statistics about a given partition
`blkstat:`	Provides statistics about a given data unit
`blkcat:`	Displays the contents of a given data unit
`blkls:`	Extracts unallocated data units from an image
`blkcalc:`	Calculates the original location where a 'blkls' data unit was found
`foremost:`	Carves out files from raw images based on the header and footer

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

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

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.

Here we see the first slide of the output from running 'fsstat' on a NTFS filesystem image.

```
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 the serial number and volume name of the image. It also shows it is NTFS from an NT, a 2000, or an 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 to 4166
Root Directory: 5
```

The second part shows the metadata information. The Master File Table (MFT) is where all 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 to 105807
Total Sector Range: 0 to 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 filesystem image has 1,024-bytes clusters and the filesystem 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 filesystem 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 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 filesystem type.

Usage:

```
blkstat [options] image
```

[Default Options Across All Sleuthkit Tools]

`-f fstype:`	Image filesystem 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 filesystem 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 filesystem the group that it is a part of.

EXAMPLE ONLY:

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

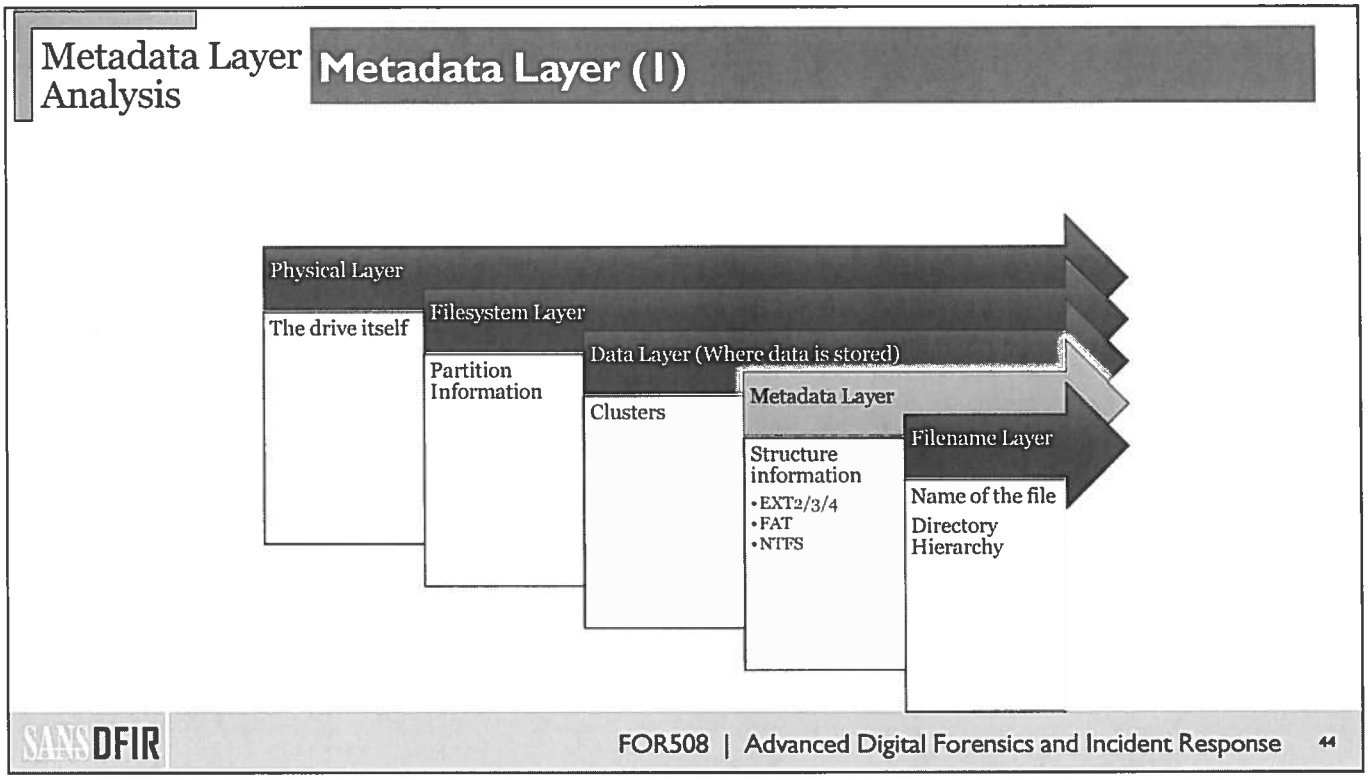
The 'blkls' tool outputs data to stdout. Its purpose is to extract the unallocated data so that deleted data could be recovered.

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 an accurate (and visual) description. blkls scans the filesystem, collecting bit from here and pieces from there. It focuses on collecting unallocated portions of the filesystem. 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.

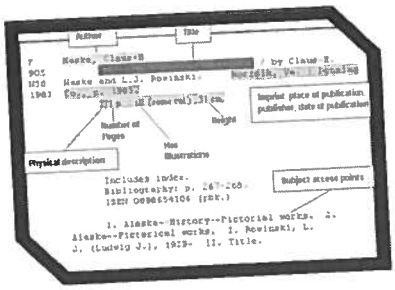


This page intentionally left blank.

Metadata Layer Analysis

Metadata Layer (2)

- The Metadata layer contains the values that describe files
- Acts like a card catalog in a library
- The metadata structures contain pointers to the Data layer and information, such as MACtimes and permissions
- Each metadata structure is given an address
- Structure names include:
  - Master File Table Entry (NTFS)
  - File Allocation Table (FAT) Directory Entry



Typical filesystems 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 filesystem. For example, it could contain a listing of directory, timestamps, and file owners.

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

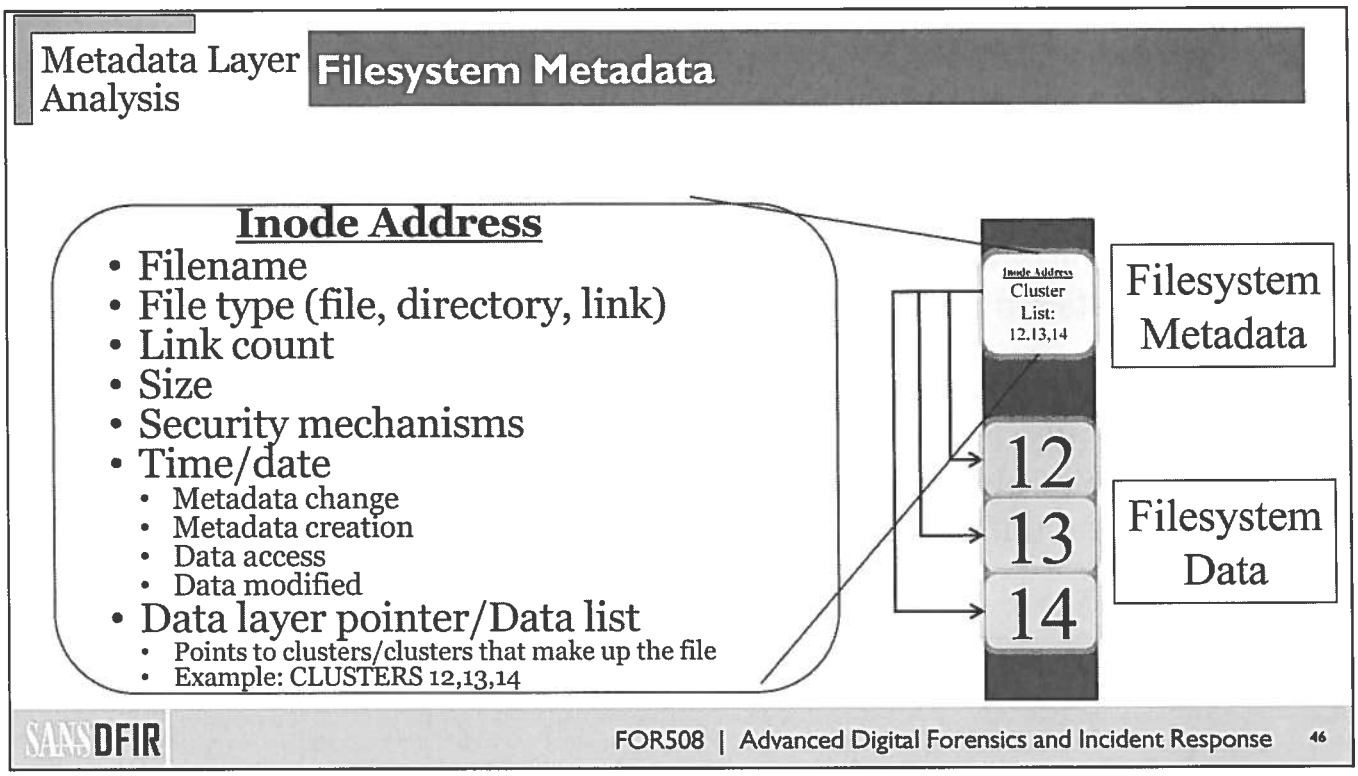
NTFS: Master File Table (MFT) entry

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 filesystems, there are a series of pointers to the different fragments. For FAT, the File Allocation Table is used to find "Chains" of clusters.

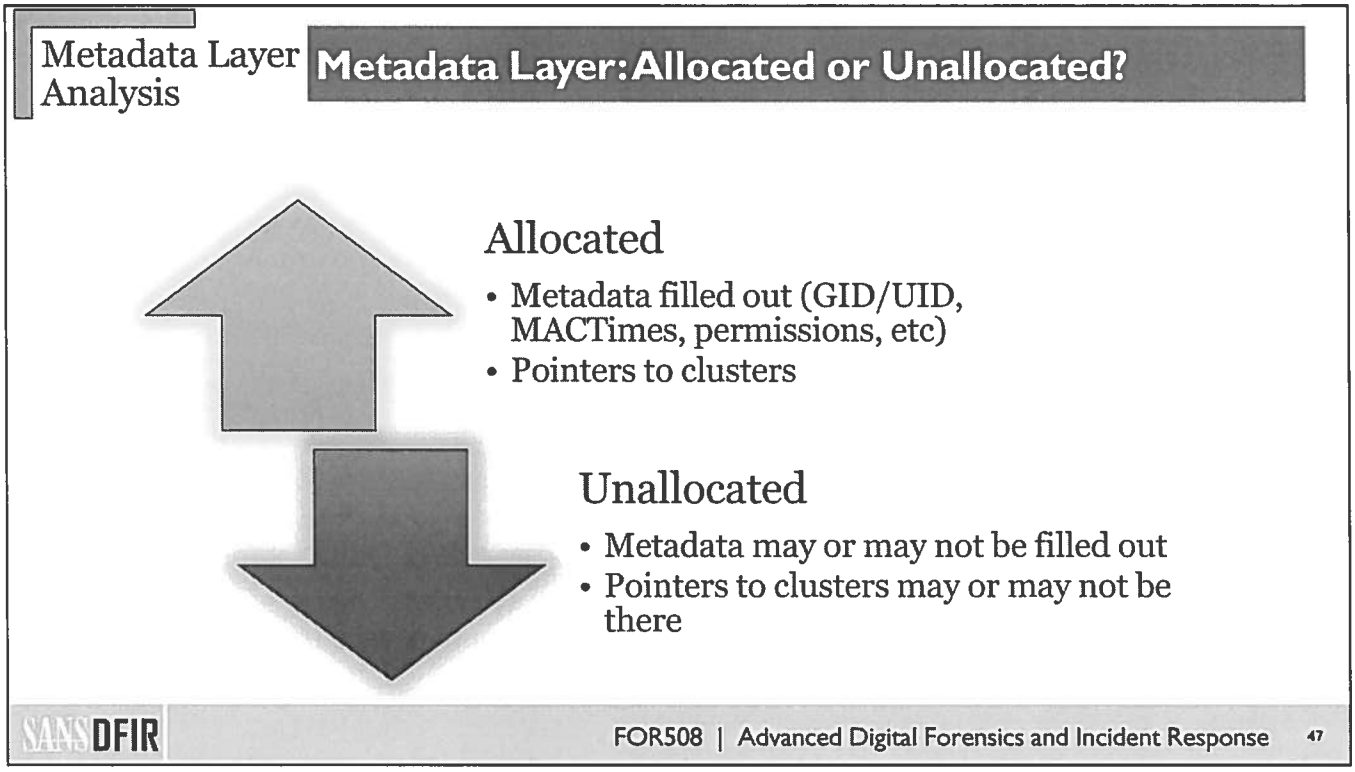
Each structure is given an address. We will use this address when referring to structures in this layer. This structure is typically hidden (especially for deleted files) from users, but there is a lot of useful information in them.

Carrier, Brian. File System Forensic Analysis. Boston, MA: Addison-Wesley, 2005. Print. Pages 186–190



The basics of any filesystem, UNIX or Windows, are similar, whereas 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 filesystem types while only the specific tool that is used changes.

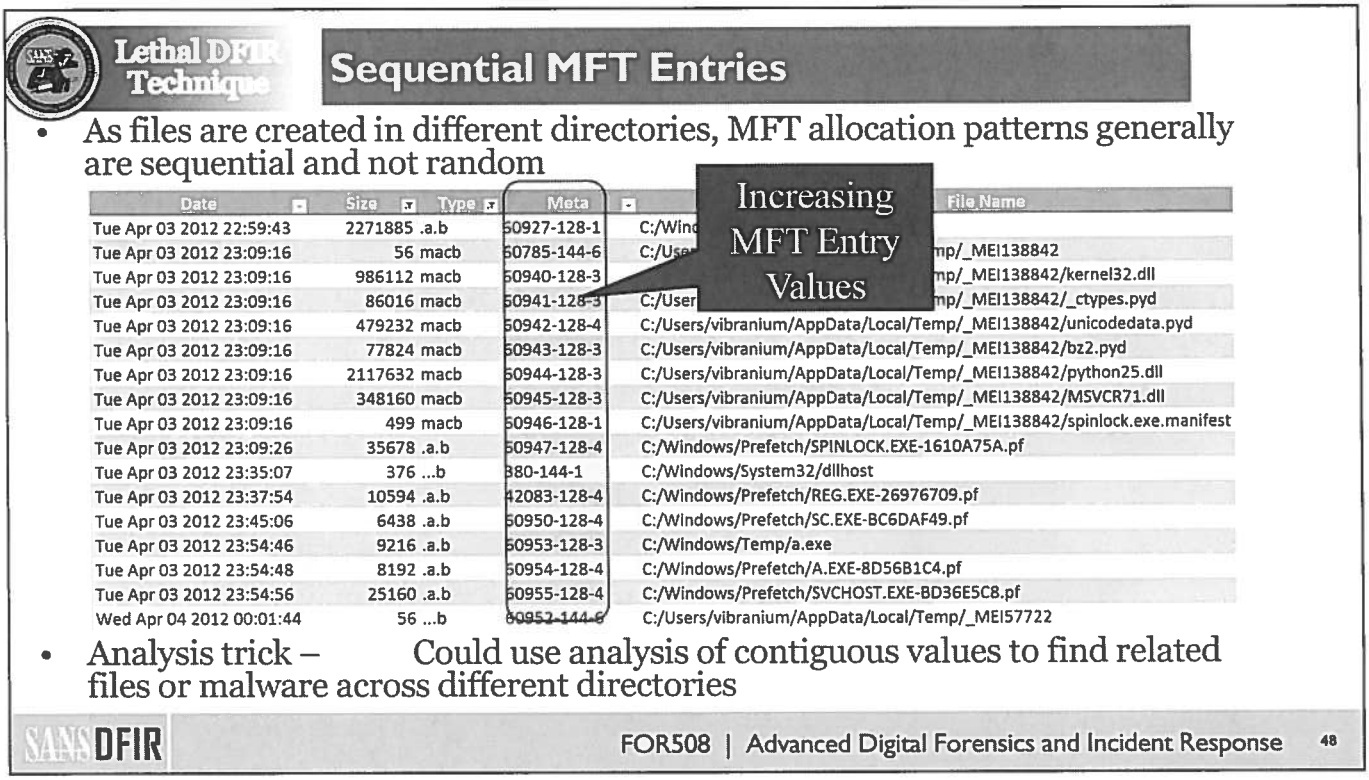
In most filesystems, elements such as the name, the type, a pointer to the data, the data size, time data, and a security mechanism will exist. Some have more robust features (NTFS and ExFAT) than others, whereas some have less (FAT).



Like data blocks/clusters, inodes are also allocated or unallocated. An allocated inode is a file that is in use by the filesystem. A file with a name points to that inode structure 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 filesystem, the data will never be filled out. When a file is deleted, the inode data is rarely wiped or overwritten.

Carrier, Brian. *File System Forensic Analysis*. Boston, MA: Addison-Wesley, 2005. Print. Page 195



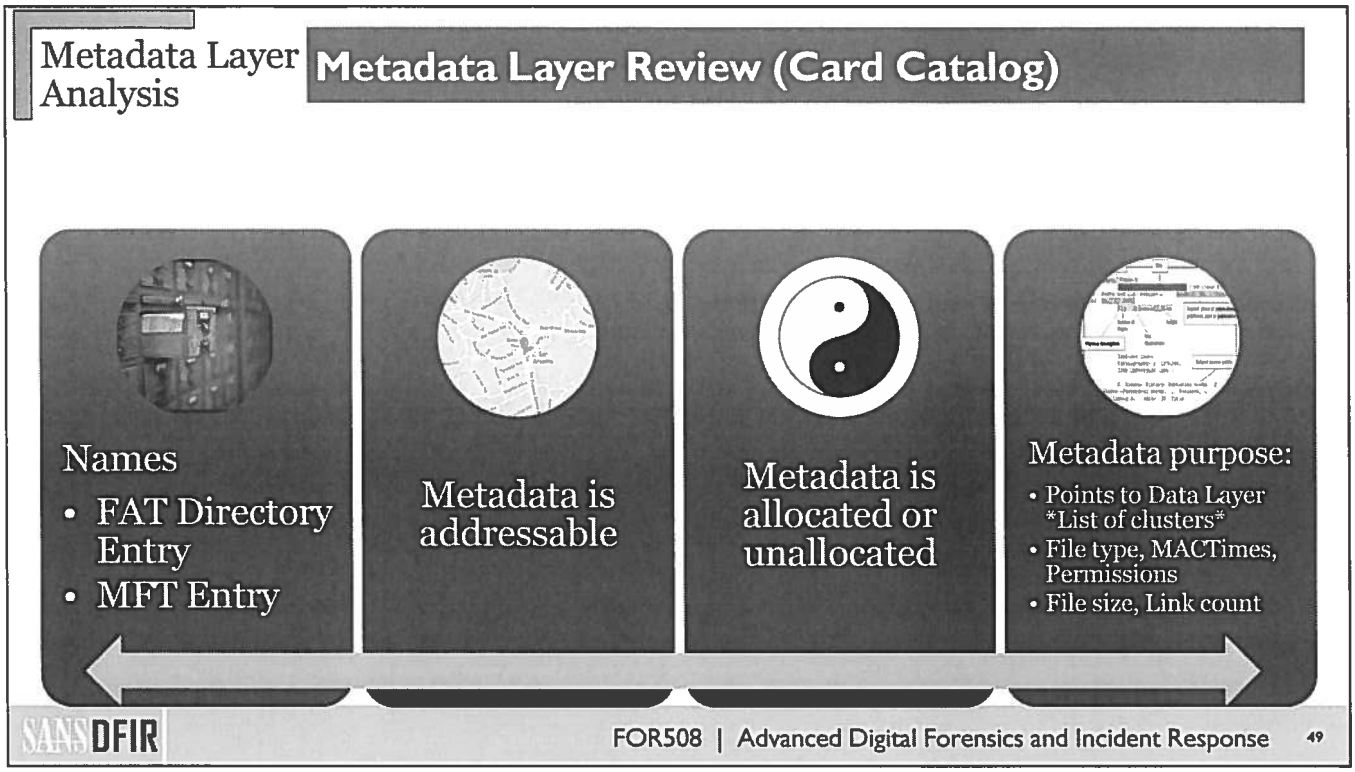
Inode and MFT entry address allocations generally are sequential in nature. This means that as new files are created on the filesystem, the following number in the inode or MFT entry allocation list (n=n+1) is generally allocated if available. In the above example, we show the initial Java Applet attack that was examined during the filesystem timeline analysis section. This is the output from the fls/mactime tool pair. We are focusing our analysis solely on files that were created (B) around the time of the attack.

As you can observe above, the sequence begins with MFT Entry 3007. While hidden by the text in the above screen shot, if you go back and examine your own timelines, you will observe that the file that was created was the C:/Windows/Sun/Java directory. As the attack created more Java files, eventually the exploit was created/dropped into the Tdungan temporary folder. The majority of the MFT entries are increasing in sequential numeric order: 3007, 3008, 3009, and so on.

Why are some of the MFT entries out of order? The answer is unknown; it is possibly a factor of processor speed and write time of the creation timestamp.

This technique is useful in determining malware footprints on a system. Certain files may be deleted later, and we might have an indicator that an older file may have existed due to the non-existent MFT entry that isn't found in the sequence. For example, we do not see the MFT entry for 3014 above. We might want to examine the MFT entry 3014 to see if the data in it were deleted or overwritten by a new file.

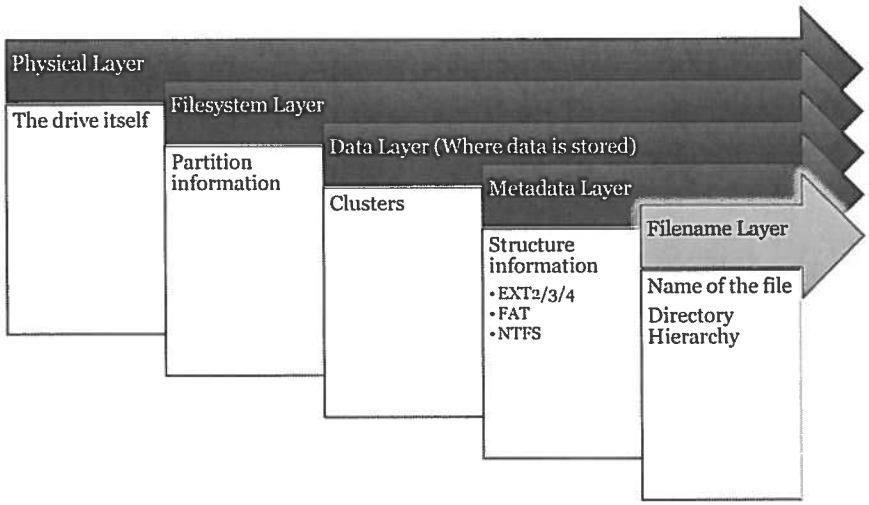
Overall, MFT and Inode allocations are likely to be sequential in nature when files are being created one after the other. If you start looking at the filesystem from this point of view, it begins to open new doors in your investigations.



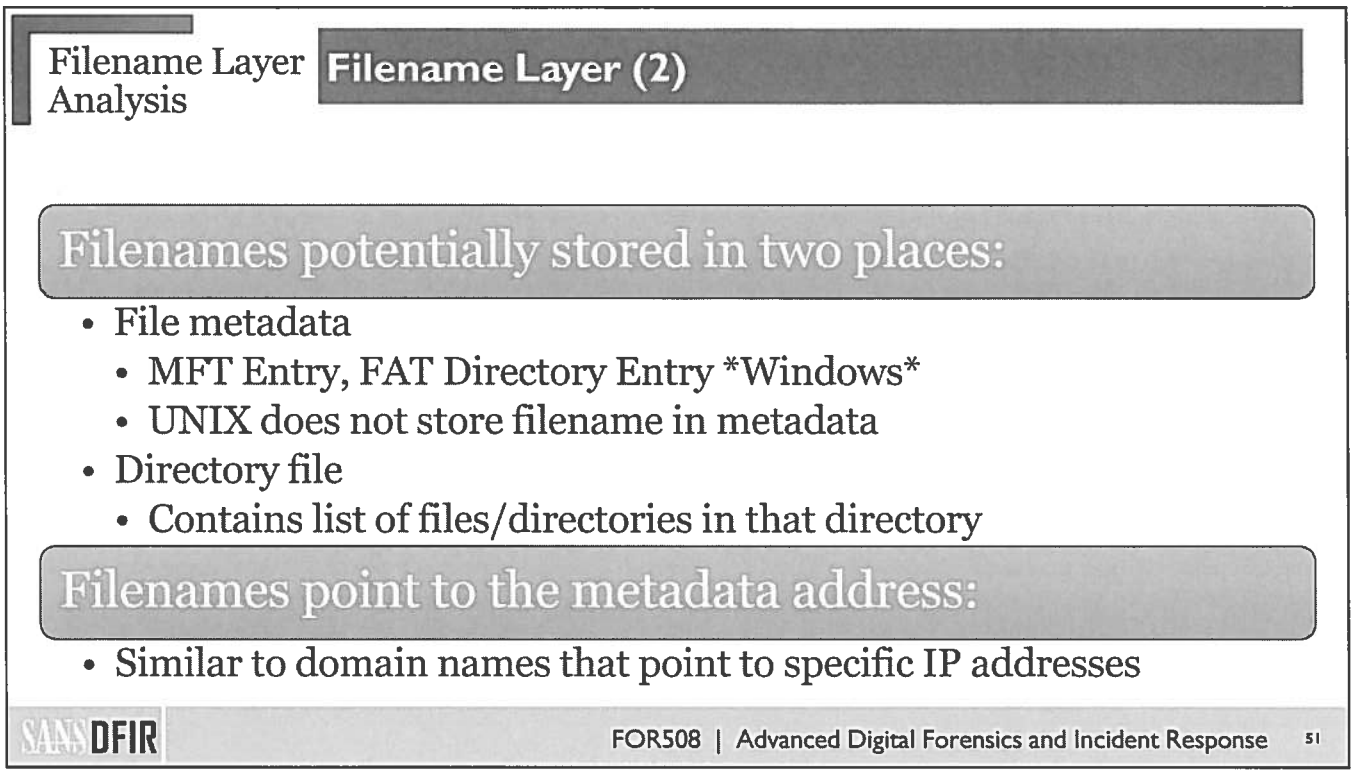
This is the essential information about the Metadata layer that would be helpful to understand.

Filename Layer Analysis

Filename Layer (1)

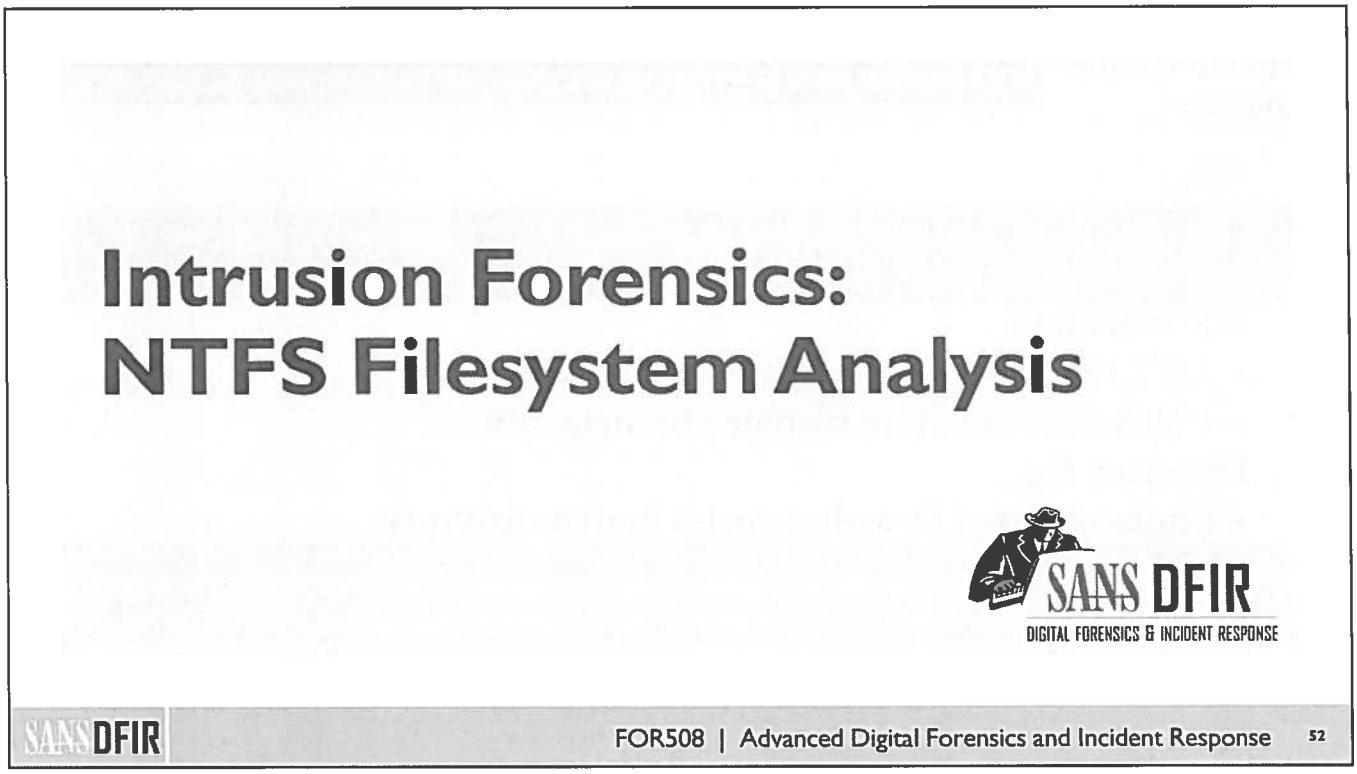


This page intentionally left blank.

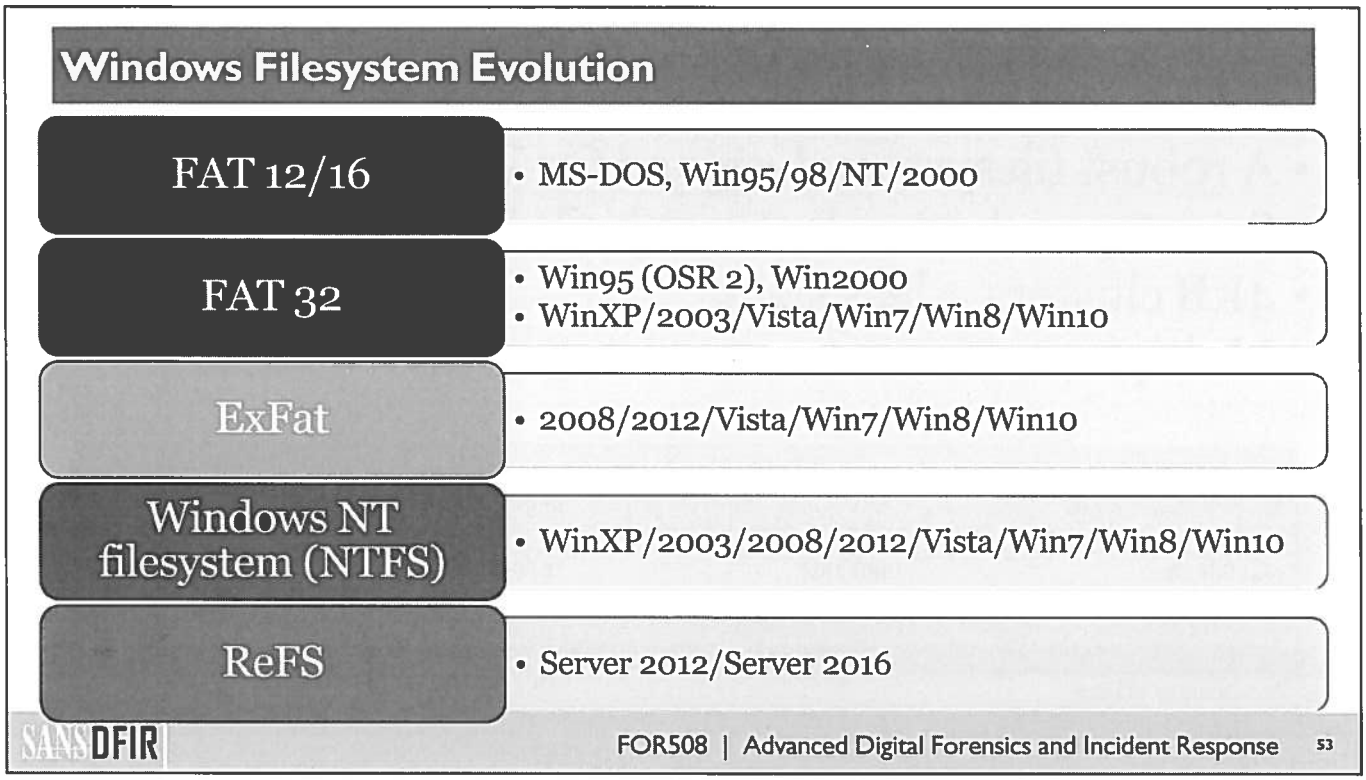


The Filename layer is typically a separate structure that gives names to files. The Metadata layer can describe everything about a file, but it is often inconvenient to have to remember that /etc/password is inode 312. The filename structures are typically stored in the data units allocated to the parent 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 filesystem will hide the filename from the user, but much data can be recovered using forensic tools.



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 filesystem. 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 filesystem has been around since the early 1980s and is one of the simplest filesystems. It contains no security features, few timestamps, and several hacks that have allowed it to still be used today. There are four variations of FAT: FAT12, FAT16, FAT32, and exFAT. The major difference in each is the size of addressable entries in the FAT, which will be described later. The exFAT filesystem 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 filesystem is one the most common PC filesystem around as it is compatible with so many different computers. FAT is reliable because it keeps a table of files and free space.

The extended FAT filesystem (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 filesystems.

NTFS Overview

- A robust filesystem 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 Filename 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)

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 filesystem, High Performance Filesystem (HPFS). NTFS was specifically designed to overcome all the shortcomings of FAT and be applicable in an enterprise environment. Therefore, it included features such as:

- Support for mixed case filenames, in different code pages
- Support for long filenames (255 characters as opposed to FAT's 8+3 characters)
- Less fragmentation of data
- Extent-based space allocation
- Transaction journaling for crash recovery
- B+ tree structure for directories
- Support for compression, encryption, and quota enforcement at the Filesystem layer
- Support for sparse files, soft links, and 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 wants.[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

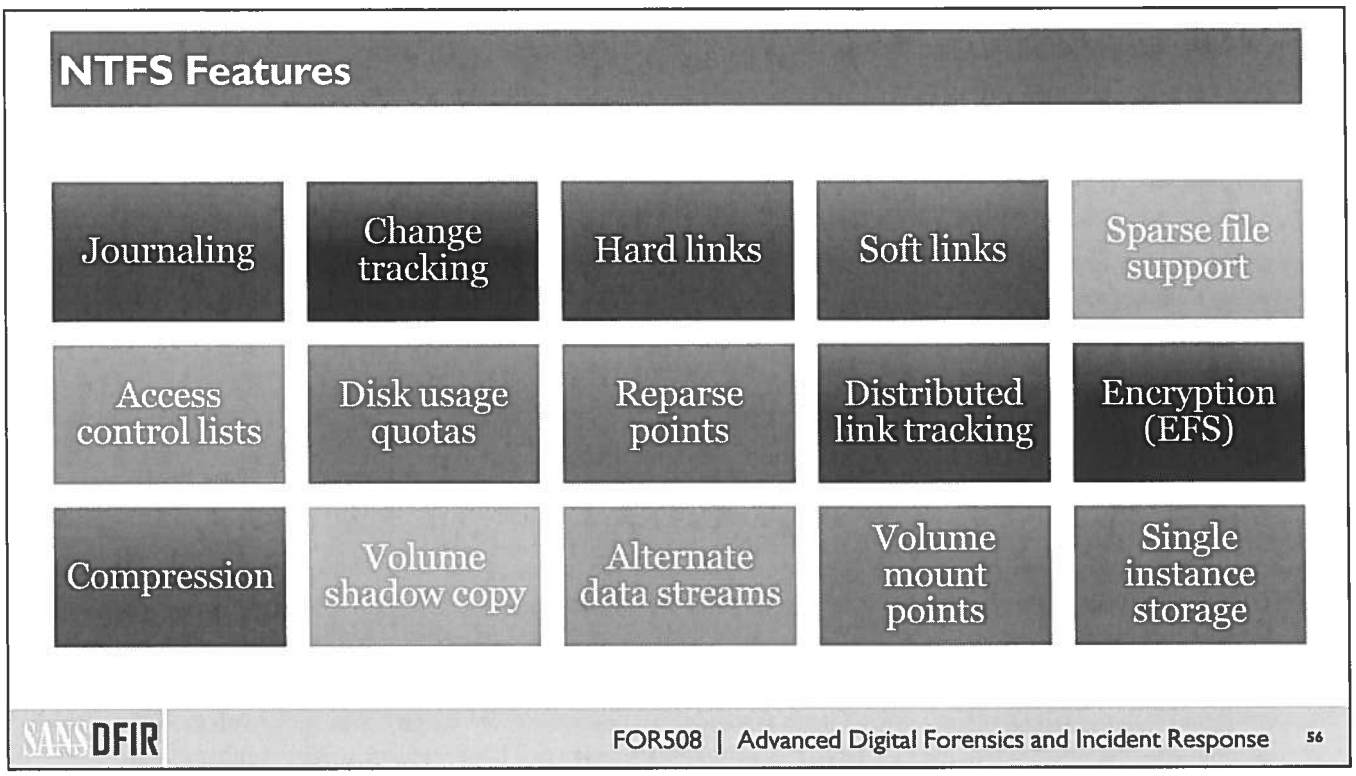
[1] http://support.microsoft.com/kb/140365

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 filesystem. 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 filesystem 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 filesystem 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 to track the state and integrity of the filesystem at all times and to correct inconsistencies causes by system crashes. Other filesystems call this "journaling," and 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. 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 filesystem 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 in 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 filesystem 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 filesystem level so that it operates completely transparent to you as the user but makes it difficult for others to read your files.

- NTFS implemented file-level compression at the filesystem 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 press 'Save' and then realize that you meant to press '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 alternative 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 alternative 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. This is 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 keeping only 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 those copies into a single copy of the actual 300GB data and a bunch of small soft links that point to that single copy, all under the hood without any of you knowing it was happening.

Metadata Layer Analysis

NTFS System Files

MFT Record #	Filename	Description
0	$MFT	Master File Table - A database that tracks every file in volume
1	$MFTMIRR	A backup copy of the first four 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 versus free) of each cluster in the volume
7	$BOOT	Boot record of the volume
8	$BADCLUS	Used to mark defective clusters so that 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 filenames
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. 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 because we are allocating space on the disk for an entire cluster, an entire cluster of MFT records will get backed up. Because the default cluster size is 4K and records are 1K, which 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 filesystem in the event of a crash. This process is called Journaling in most other filesystems 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 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, respectively.

$BOOT

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

$BADCLUS

This file provides the filesystem 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. Because 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 an 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 next.

$UPCASE

This file contains a table of uppercase and lowercase unicode letters for each unicode code page in use for filenames within the system. This table is used in sorting the files by name so that "A" and "a" are next to 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 files were placed into normal records for regular use. Because the files below are 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). 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

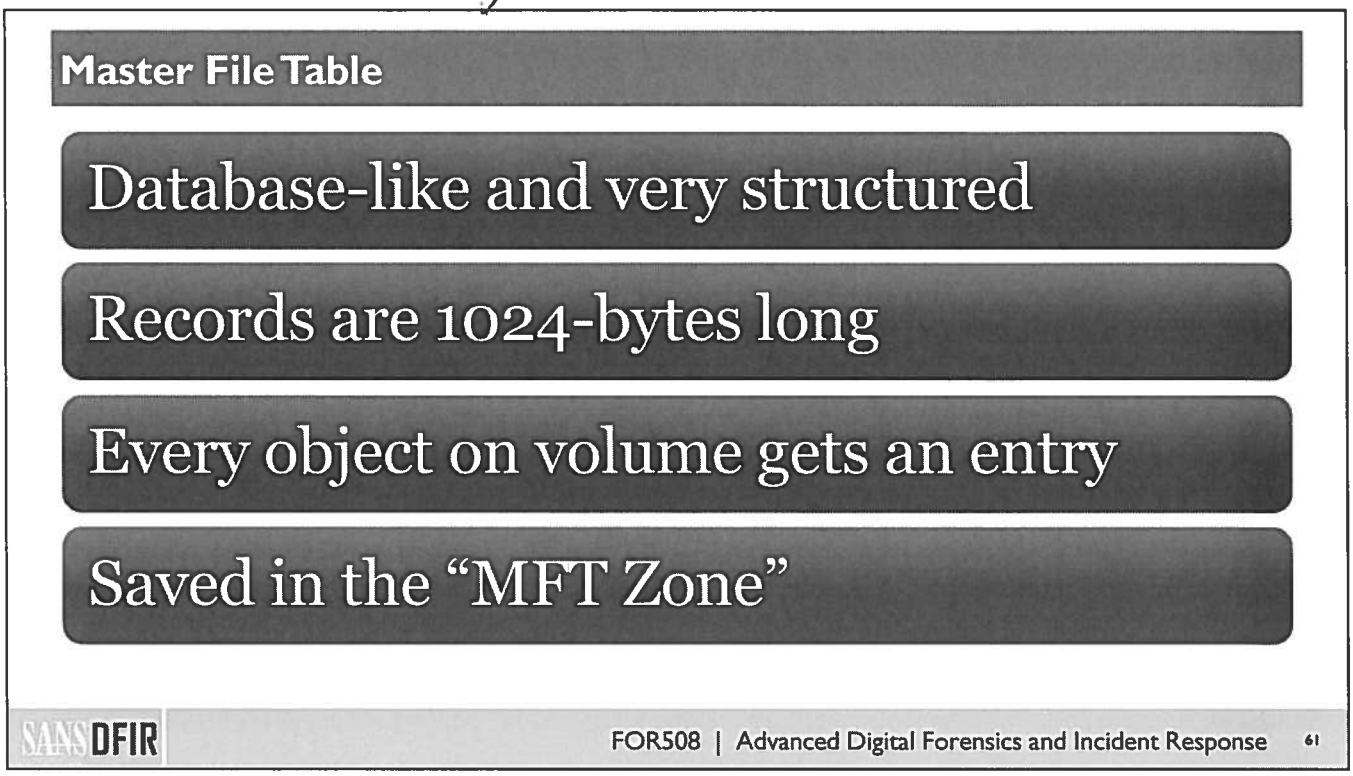
This file contains an index of all the reparse points on the volume. Reparse points have a multitude of uses, but the most common is for symbolic links, in which a file is really just a pointer to another file and editing this file is really modifying the file it points to. Reparse points are also used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

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



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 a 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.

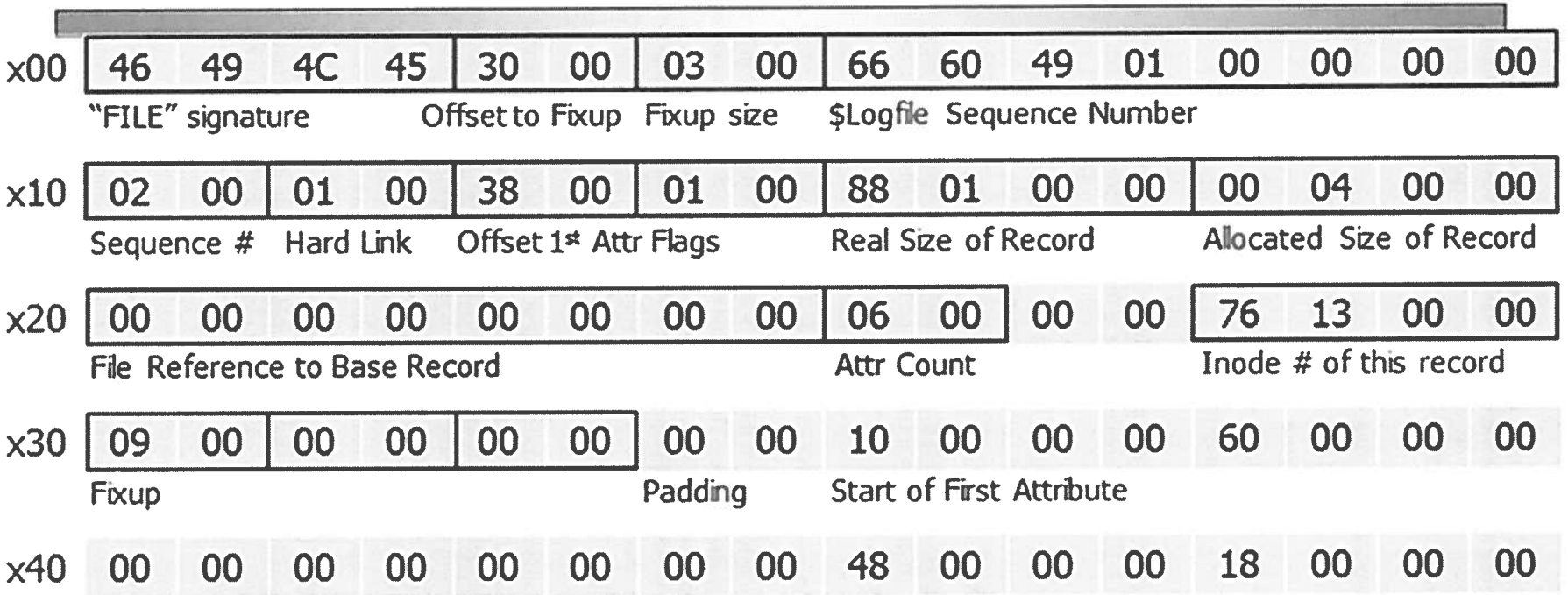
Following the signature field is a 2-byte value that gives the offset to the Fixup Array relative to the start of the MFT entry and a second 2-byte value that tells the number of entries in the Fixup Array. The Fixup Array is used for error checking for MFT data structures that span multiple sectors, excluding sectors that contain file data. When these data structures are written to disk, the last 2 bytes of each sector are copied into a special array, and a signature value is written to the last 2 bytes of each sector. When the data structures are read by the system, the last 2 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 filesystem and is used to determine whether the filesystem 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 reused. When an MFT record is allocated for the first time, its Sequence Number will be set to 1, when the file is deleted, and the MFT record is unallocated, the Sequence Number is incremented again and will be incremented only 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



The Hard Link count at offset x12 refers to the number of $File_Name attributes there are for this file. Multiple filename attributes could be long and short filenames 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 used only for Extended Records, which are used only when a record contains so many attributes that it can't fit them all within 1,024 bytes. So, for the vast majority of records, this will be all zeros.

The Attribute Count field at offset x28 essentially tells 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 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 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 an x2A in older versions to an 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 2-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 2-byte entries make up the USN+USA. At the end of each 512-byte sector in the 1,024-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 2 bytes that should be in the last 2 bytes of each sector of the record. So, when reading in the first sector of the record, the system checks the last 2 bytes against the USN and then replaces those 2 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

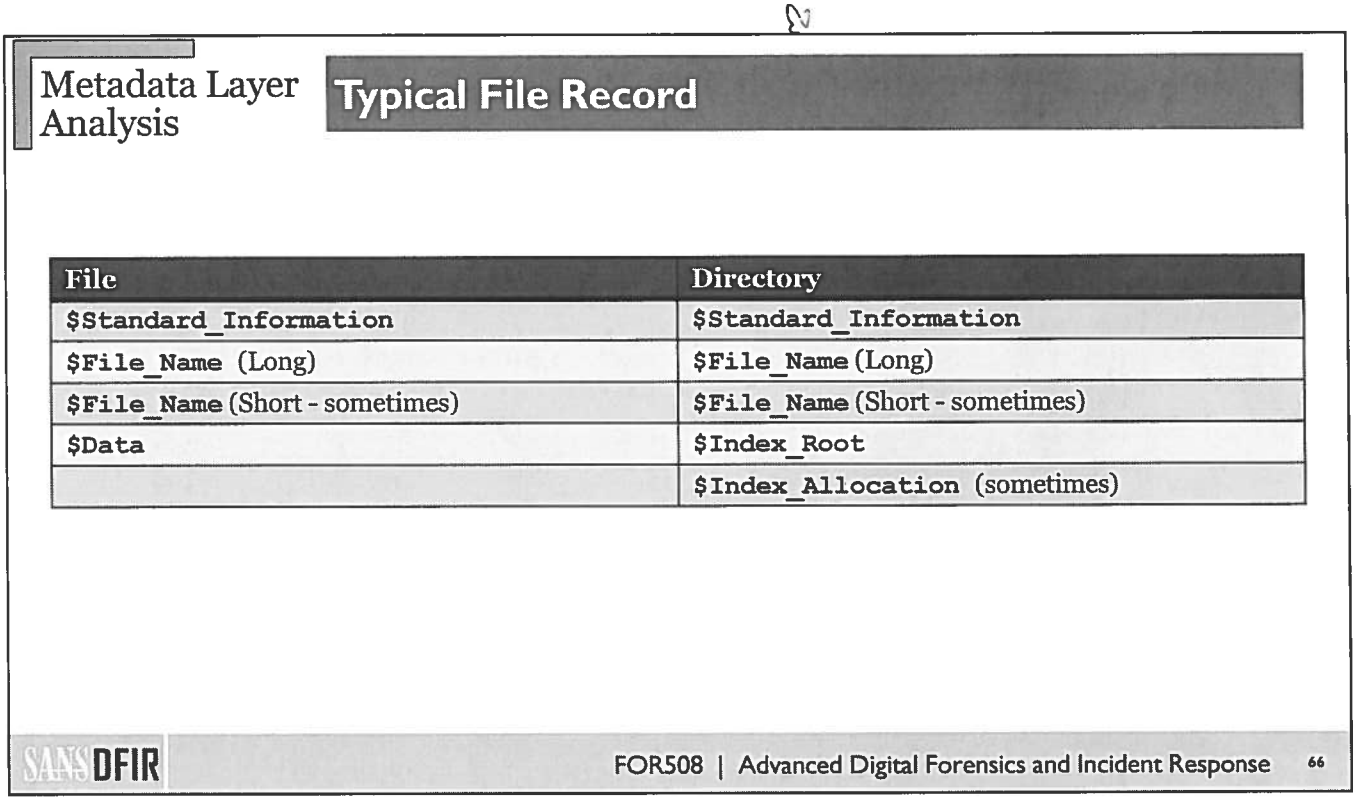
NTFS Attributes

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 timestamps, 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, timestamps, 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, and so on.
$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 reparse tag.
$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.



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

If the name is longer than eight 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 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. When 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 show up only 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.

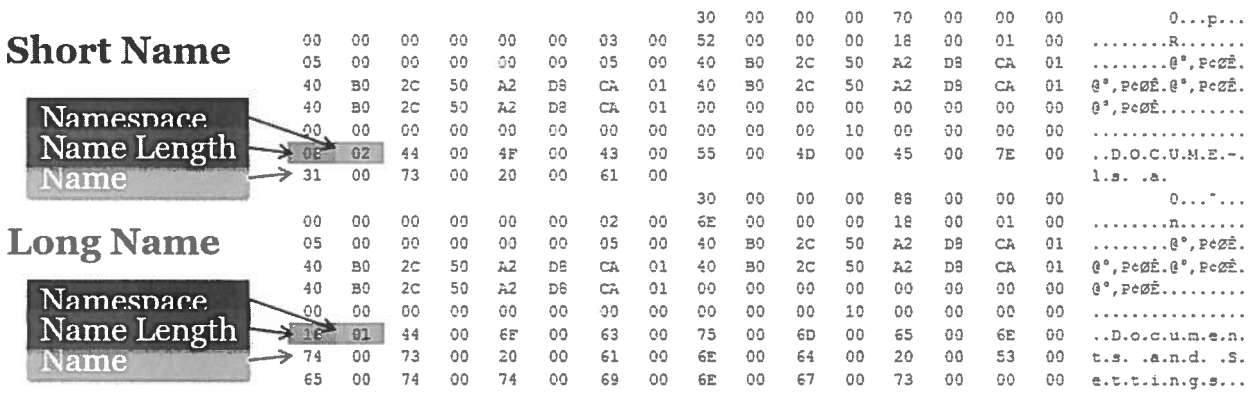
In the next few pages we will examine the attributes above in more detail. We have a lot to cover this week, so we are just going to cover the most forensically important attributes.

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 three different sets of timestamps for the file. As we will see later, entries in directory listings follow the same format as above, giving us yet another set of timestamps for the same file.

Immediately preceding the name are two 1-byte fields. The first one tells us the number of 2-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 need only one $File_Name.



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 filesystem to display the time information. Other timestamps will be kept in other locations that we will see later, but these are the definitive set.

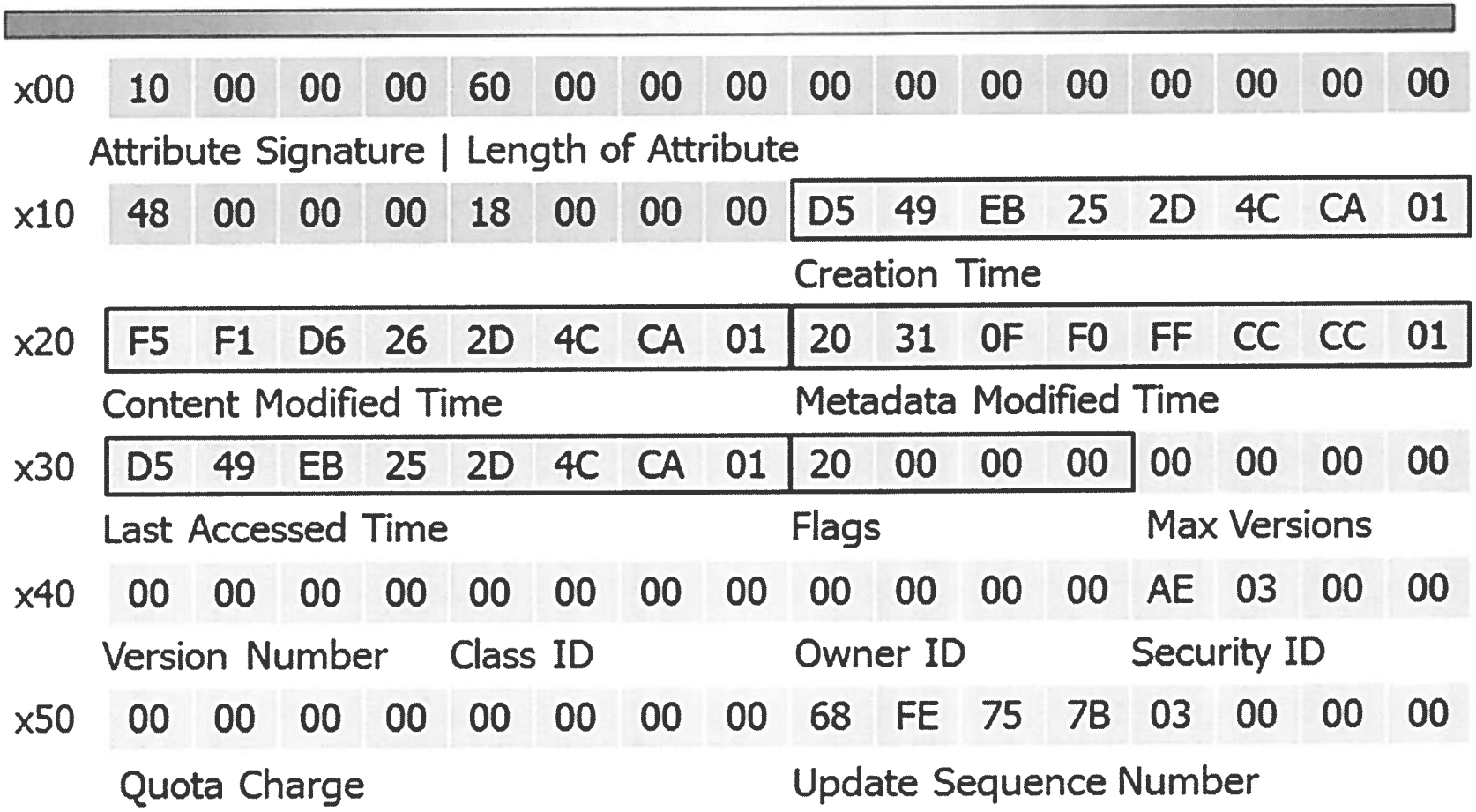
At offset x38 will be a set of flags. These flags track the file attributes, such as hidden, read only, and so on.

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

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 filesystem. (The Security ID here has no relation to the SID used in Windows to track the user.) They are vitally important when parsing the $Secure and $UsnJrnl as some entries in that index will refer to only the file by the tracking number here. Most of these fields are present only in version 3.0 and 3.1.

Carrier, Brian. *File System Forensic Analysis*. Boston, MA: Addison-Wesley, 2005. Print. Pages 360–362.

$Standard_Information



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															

SANS DFIR FOR508 | Advanced Digital Forensics and Incident Response 70

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 2 bytes are a sequence number for that MFT Record.

Though the $File_Name attribute contains the same four timestamps as $Standard_Information, these are updated only if this attribute is changed, such as by renaming or moving a file. These timestamps are hard to modify naturally using the win32API. During timeline analysis using these values, 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 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 timestamps 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.

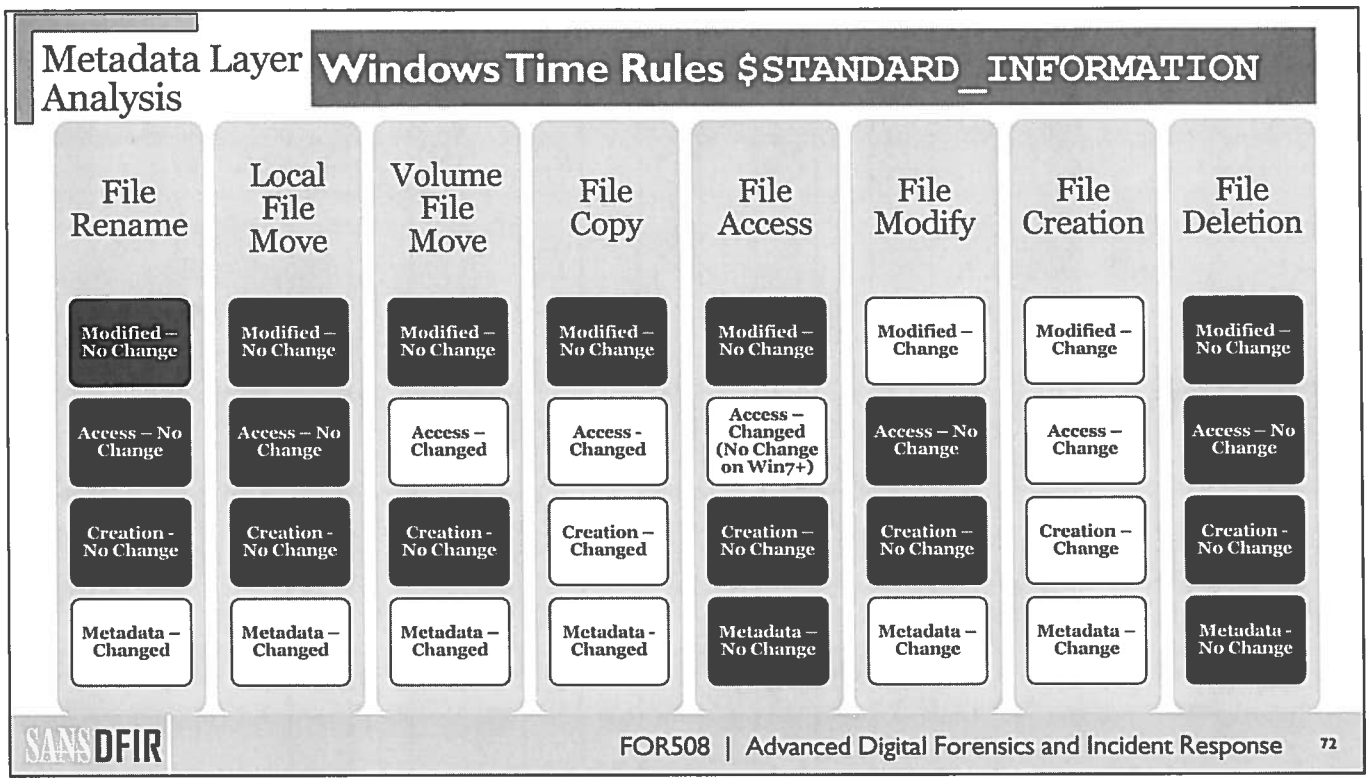
Each attribute must start at an increment of 8 bytes, so there will be 0 to 7 bytes of padding after the filename.

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

[2] https://github.com/dkovar/analyzeMFT

$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															

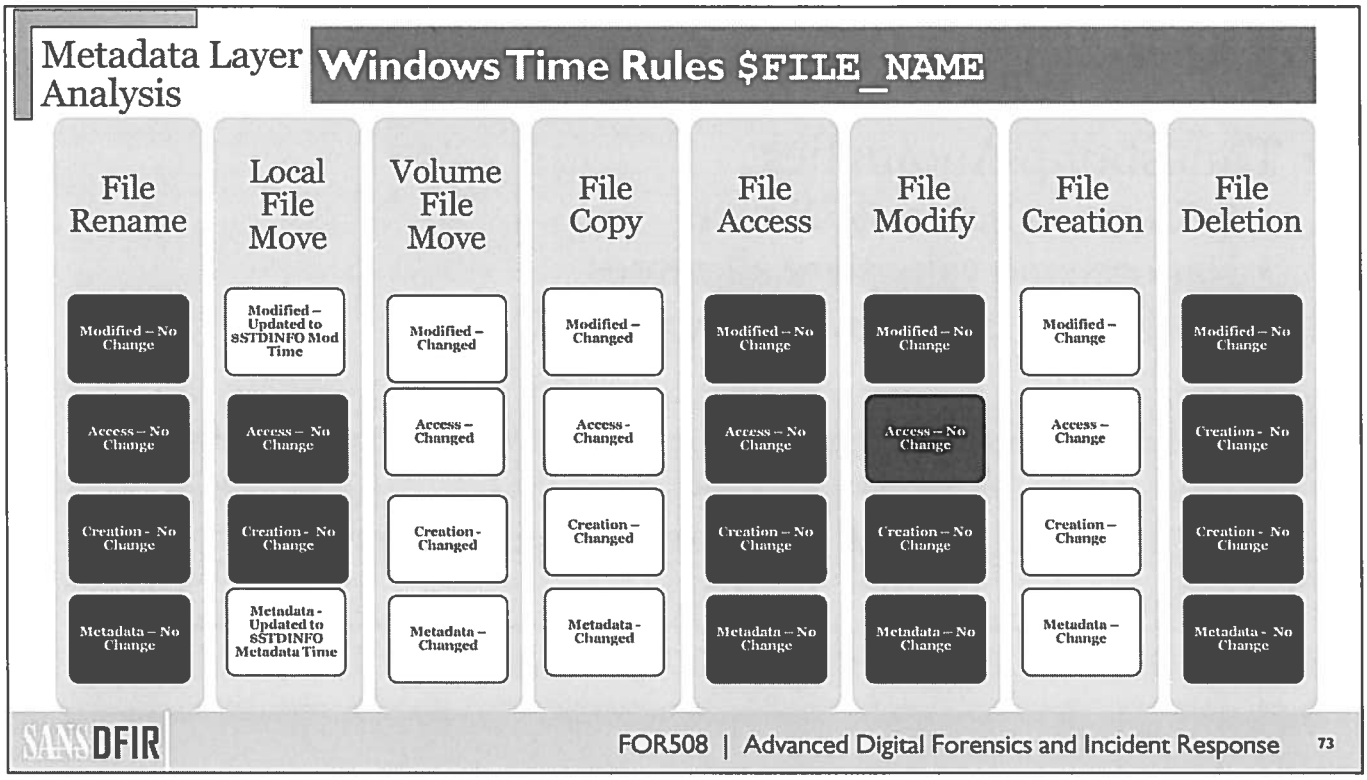


These are the general rules when it comes to files being moved, copied, accessed, modified, or created. The MFT change time is still being researched in regards to all of these times.

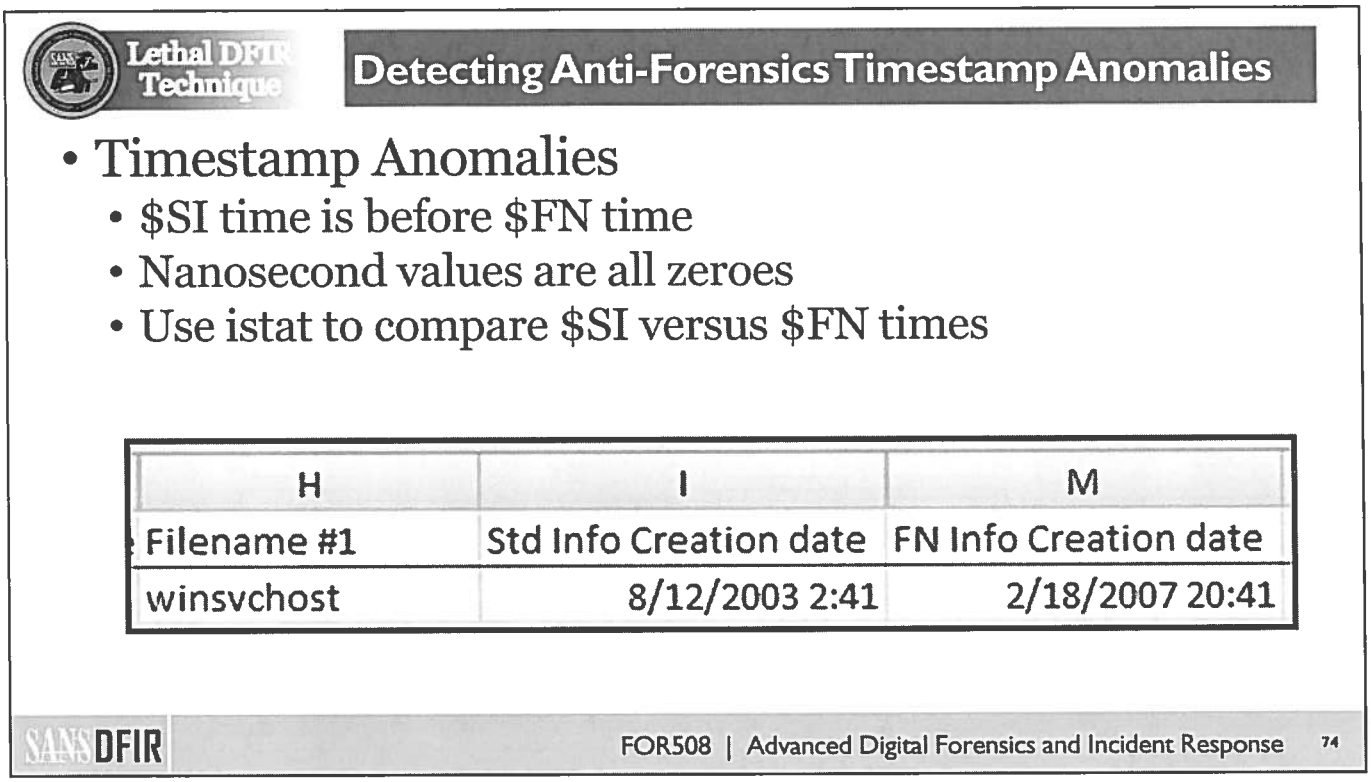
One note: When referring to the move command, we are generally referring to moving a file via "Cut-and-Paste" and not a move via a command line.

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 responders 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.

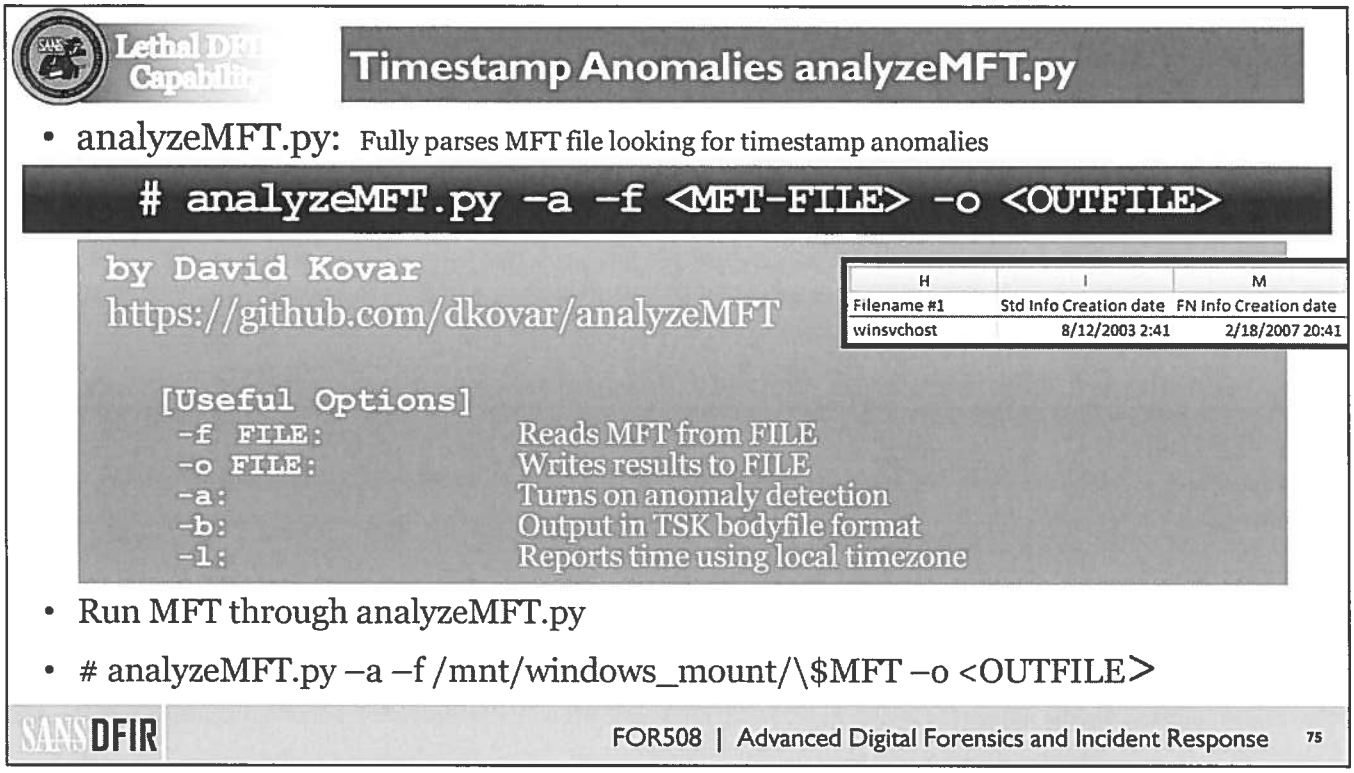


This is the same tests performed while examining the $FILENAME timestamps. The data here could help you determine when potential anti-forensics backdating of file timestamps might have occurred. There might be more than a single $Filename attribute per file. This would translate into an additional four 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.



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 hackers 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 responder 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

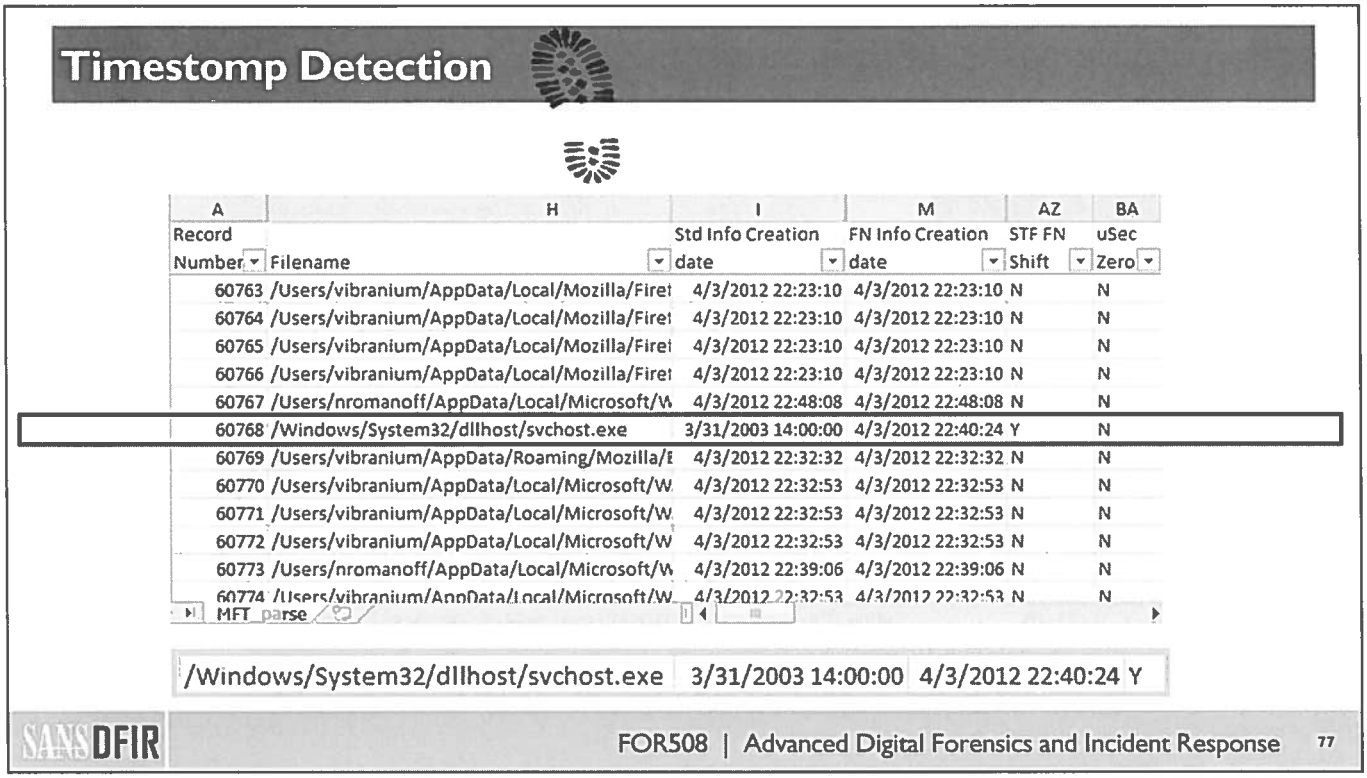
- Record Sequence - The sequence number for the record
- Parent Folder Record Number
- Parent Folder Sequence Number
- For the standard information attribute:
    - Creation date
    - Modification date
    - Access date
    - Entry date
- For up to four filename records:
    - filename
    - Creation 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:

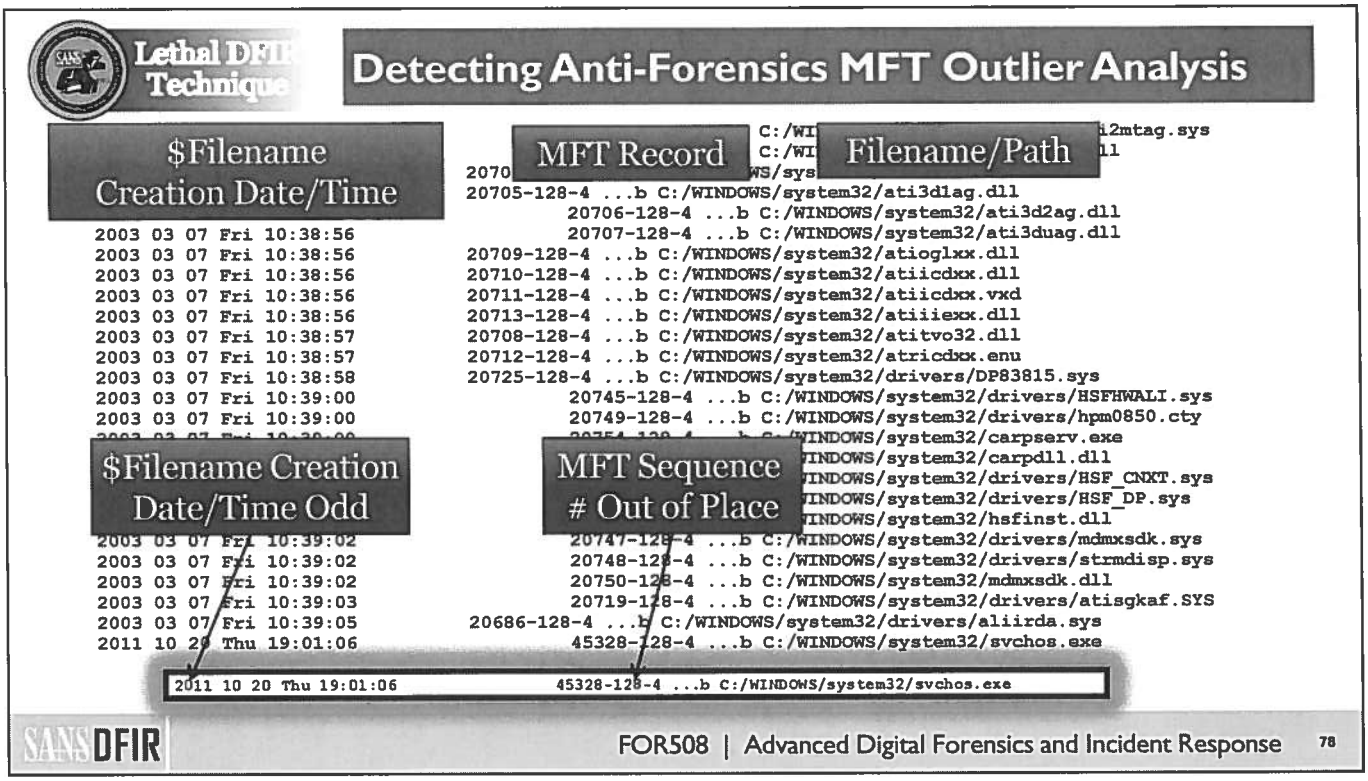
> Standard Information, Attribute List, Filename, Object ID, Volume Name, Volume Info, Data, Index Root, Index Allocation, Bitmap, Reparse Point, EA Information, EA, Property Set, Logged Utility Stream

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.



This page intentionally left blank.



On NTFS the first file on the filesystem 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 1, and so on. On EXT2/3/4 filesystems, the root directory (/) is assigned MFT Record Number 2 and things increase from there as additional files are stored on the system.

A typical filesystem 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, and so on, but MFT Record Numbers contain owner, group, location on disk, size of file, and such. In a library these cards are arranged alphabetically either by title, author, or subject. In a filesystem, they are simply first come, first serve and are numbered sequentially. In NTFS MFT Record Number 0 always points to the $MFT.

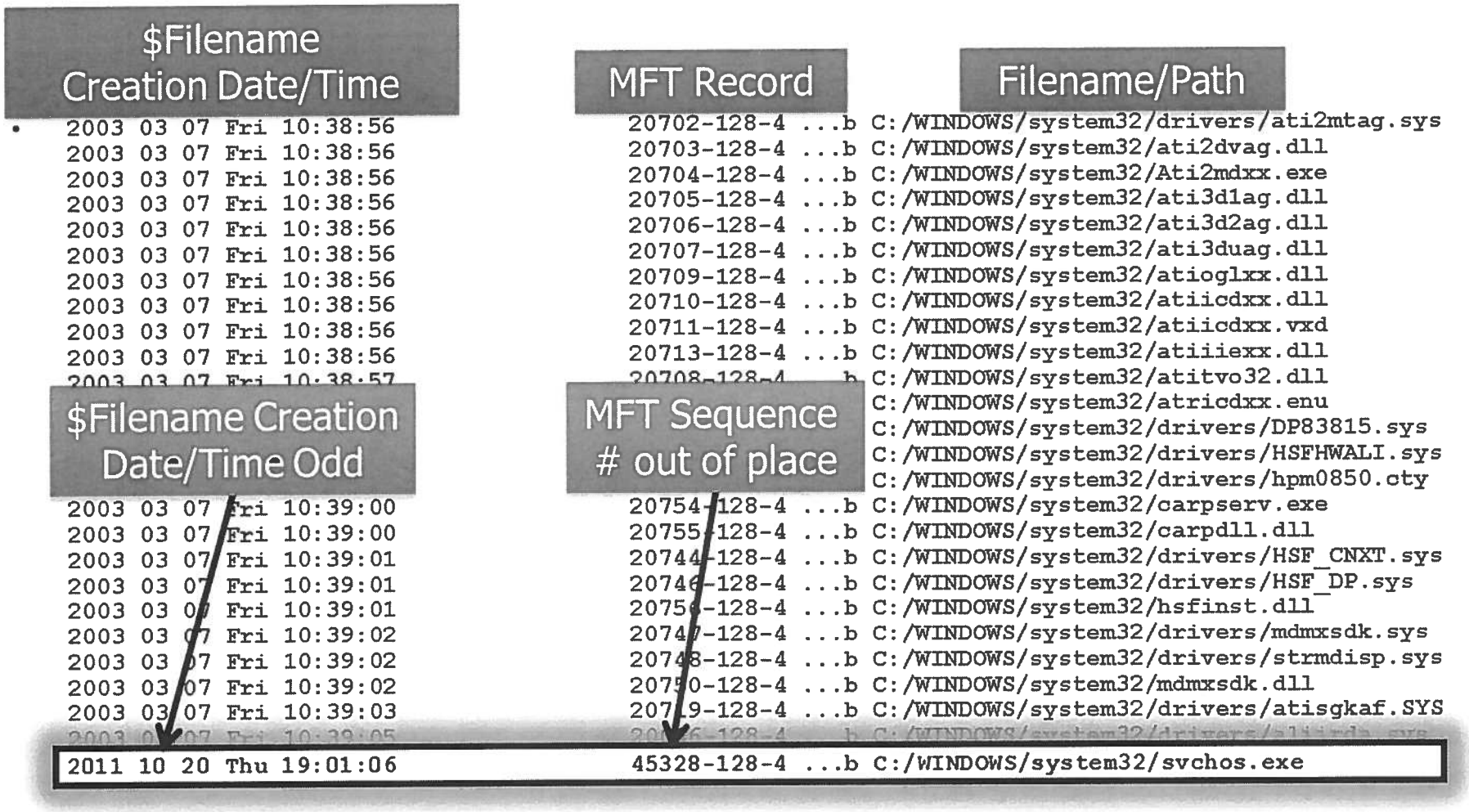
Given that these MFT Record Numbers are assigned sequentially, if new files are written to disk, the MFT Record Numbers that are assigned to them are likely to be sequential or close to sequential, assuming a sequential run of MFT Record Numbers is available. I need to hire a good illustrator to animate 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 1 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 filesystems 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 an 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. But this technique is not well known or widely used. As a result few hackers have implemented any anti-forensics capability to thwart this technique as they are not thinking about the way the filesystem will perform MFT Record Number allocation.


$Filename Creation Date/Time
MFT Record
Filename/Path
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
2003 03 07 Fri 10:38:56
20702-128-4 ...b C:/WINDOWS/system32/drivers/ati2mtag.sys
20703-128-4 ...b C:/WINDOWS/system32/ati2dvag.dll
20704-128-4 ...b C:/WINDOWS/system32/Ati2mdxx.exe
20705-128-4 ...b C:/WINDOWS/system32/ati3d1ag.dll
20706-128-4 ...b C:/WINDOWS/system32/ati3d2ag.dll
20707-128-4 ...b C:/WINDOWS/system32/ati3duag.dll
20709-128-4 ...b C:/WINDOWS/system32/atioglxx.dll
20710-128-4 ...b C:/WINDOWS/system32/atiicdxx.dll
20711-128-4 ...b C:/WINDOWS/system32/atiicdxx.vxd
20713-128-4 ...b C:/WINDOWS/system32/atiiiexx.dll
$Filename Creation Date/Time Odd
MFT Sequence # out of place
2003 03 07 Fri 10:39:00
2003 03 07 Fri 10:39:00
2003 03 07 Fri 10:39:01
2003 03 07 Fri 10:39:01
2003 03 07 Fri 10:39:01
2003 03 07 Fri 10:39:02
2003 03 07 Fri 10:39:02
2003 03 07 Fri 10:39:02
2003 03 07 Fri 10:39:03
2011 10 20 Thu 19:01:06
45328-128-4 ...b C:/WINDOWS/system32/svchos.exe

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															

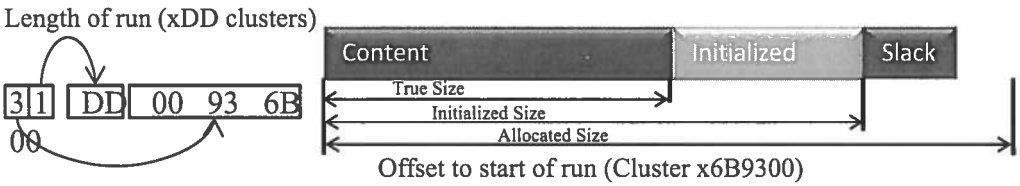
SANS DFIR FOR508 | Advanced Digital Forensics and Incident Response 81

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.

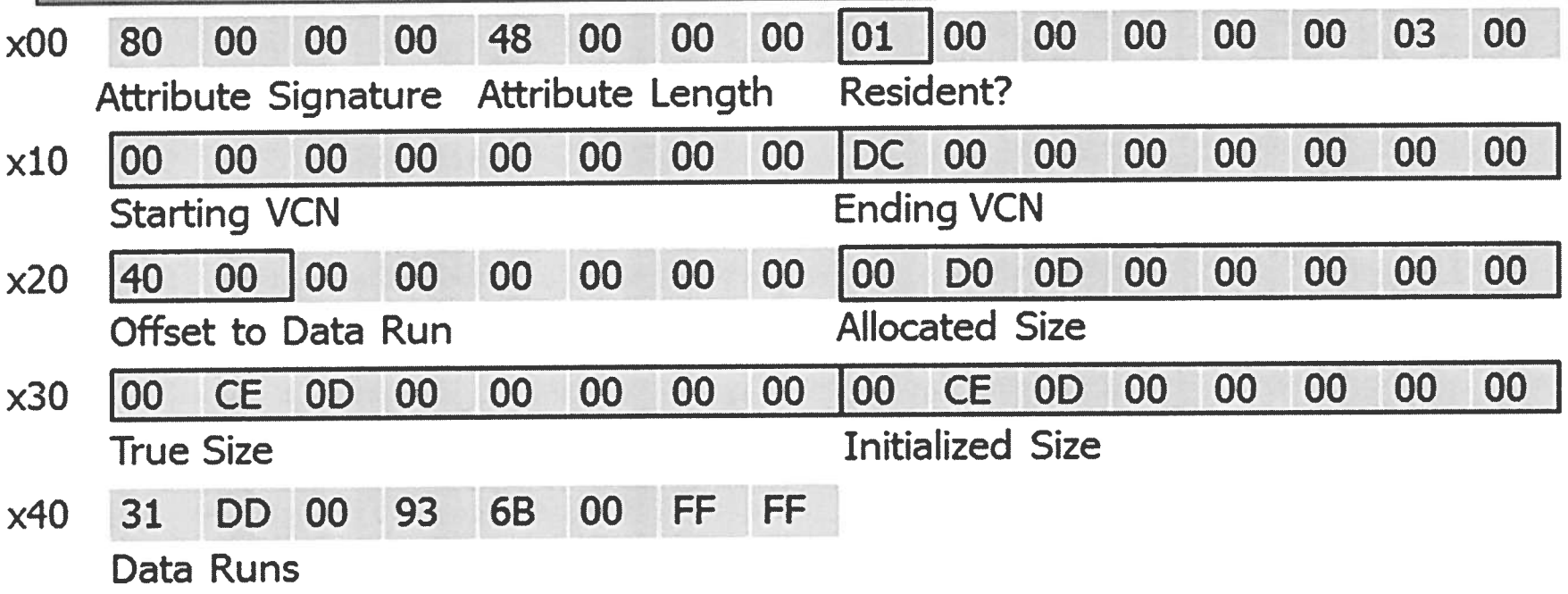
Data Runs consist of a series of entries that tell us the starting point and length of each segment of the file. Each entry has a 1-byte header that consists of two fields, one in each nibble. The 31 from the example above tells us that we need to read in 1 byte to get the length of the run and 3 bytes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

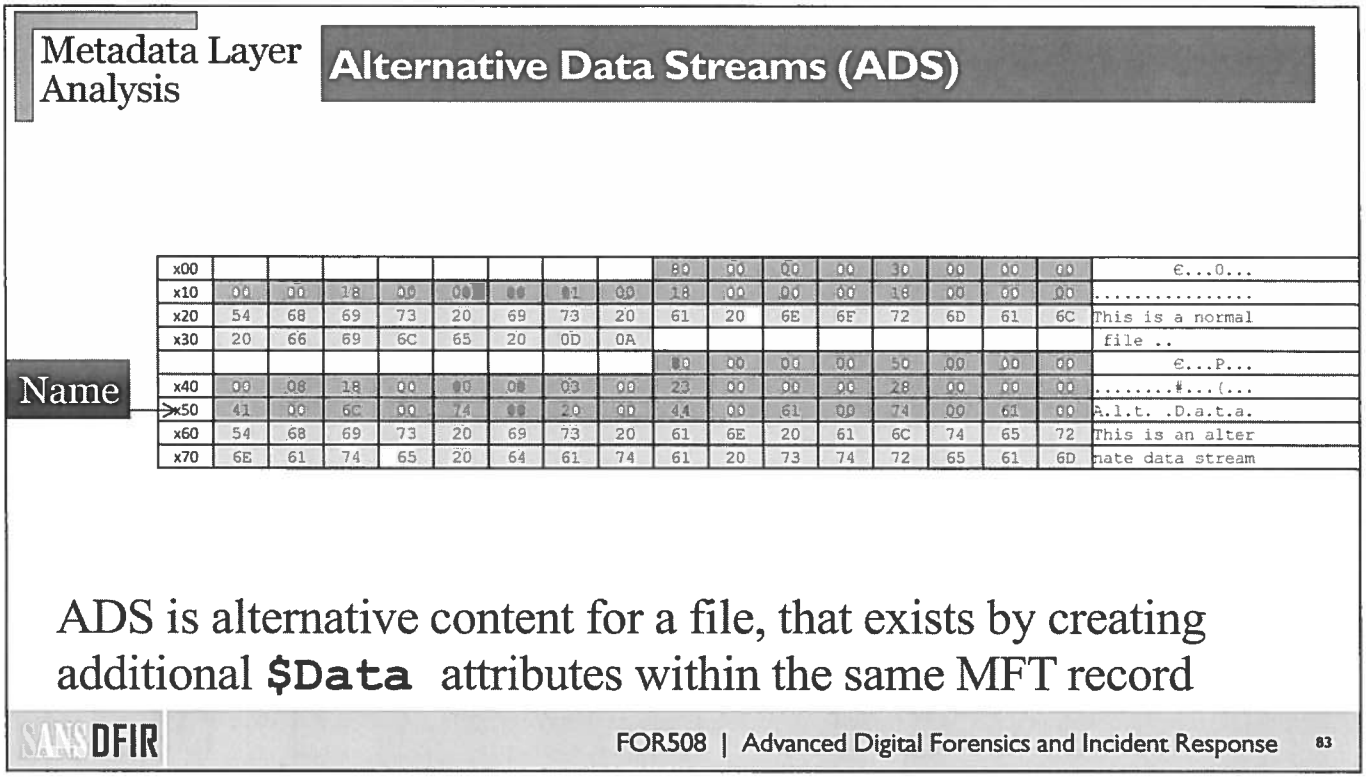


Carrier, Brian. *File System Forensic Analysis*. Boston, MA: Addison-Wesley, 2005. Print. Pages 355–359.

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

$Data





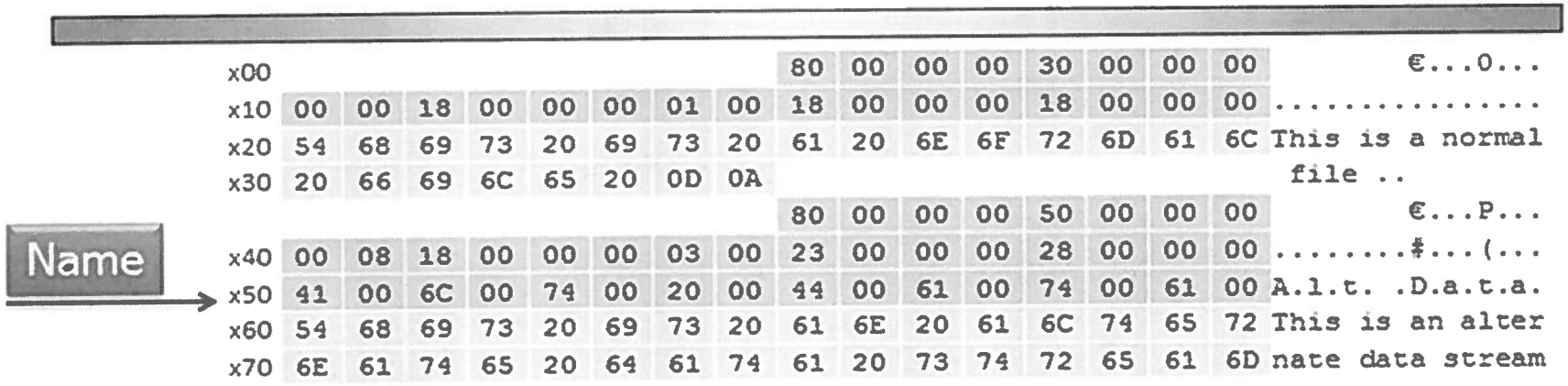
'cause sometimes one just isn't enough.

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 filesystems. 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.

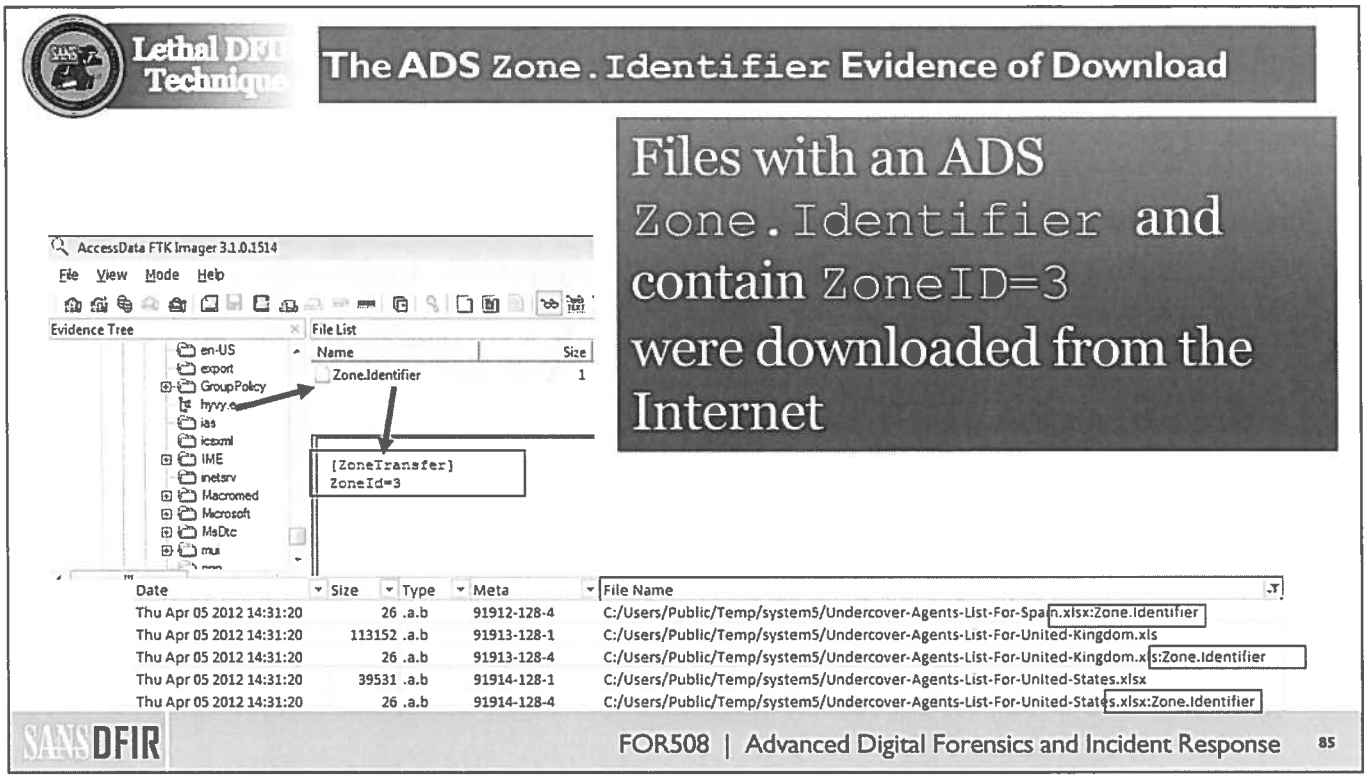
Alternative Data Streams are merely the presence of a second or subsequent $Data attribute, much in the same vein that a file can have multiple $File_Name attributes. The primary attribute is never named, deriving its name from the $File_Name attribute. Any subsequent $Data attributes must be named to be 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.

Because the rest of Windows is designed only to accommodate the primary data stream, most tools cannot report on the existence of an alternative data stream, let alone its size and contents. No other metadata about an alternative 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



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



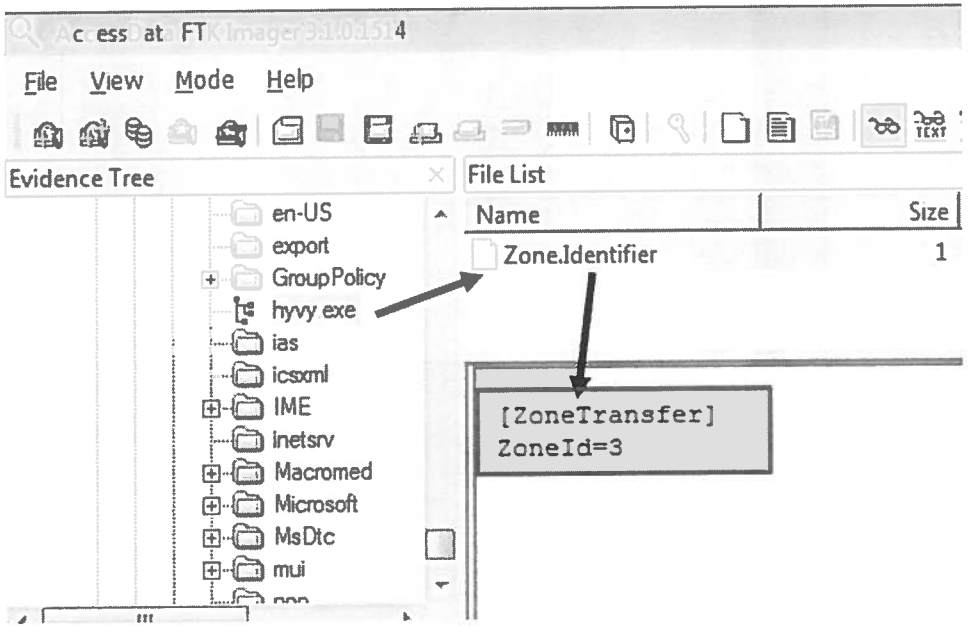
Starting with XP SP2 when files are downloaded from the Internet via a browser to a NTFS volume, an alternative data stream is added to the file. The alternative data stream is named "Zone.Identifier." The Zone.Identifier usually has simple text with the stream "ZoneID=3." Value 3 means that the file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

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.

Above you can easily view the zone.identifier using FTK Imager to see the alternative data stream. In addition, The Sleuth Kit Tool fls also extracts alternative data streams such as these.



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

The Sleuth Kit MFT Analysis Tools

ifind:	Finds the metadata address of a given cluster
istat:	Lists entry data at a specific metadata address
icat:	Extracts data block pointed to from metadata

In the next few slides, we will cover some of the key metadata 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 filename
            -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 filesystems, 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]

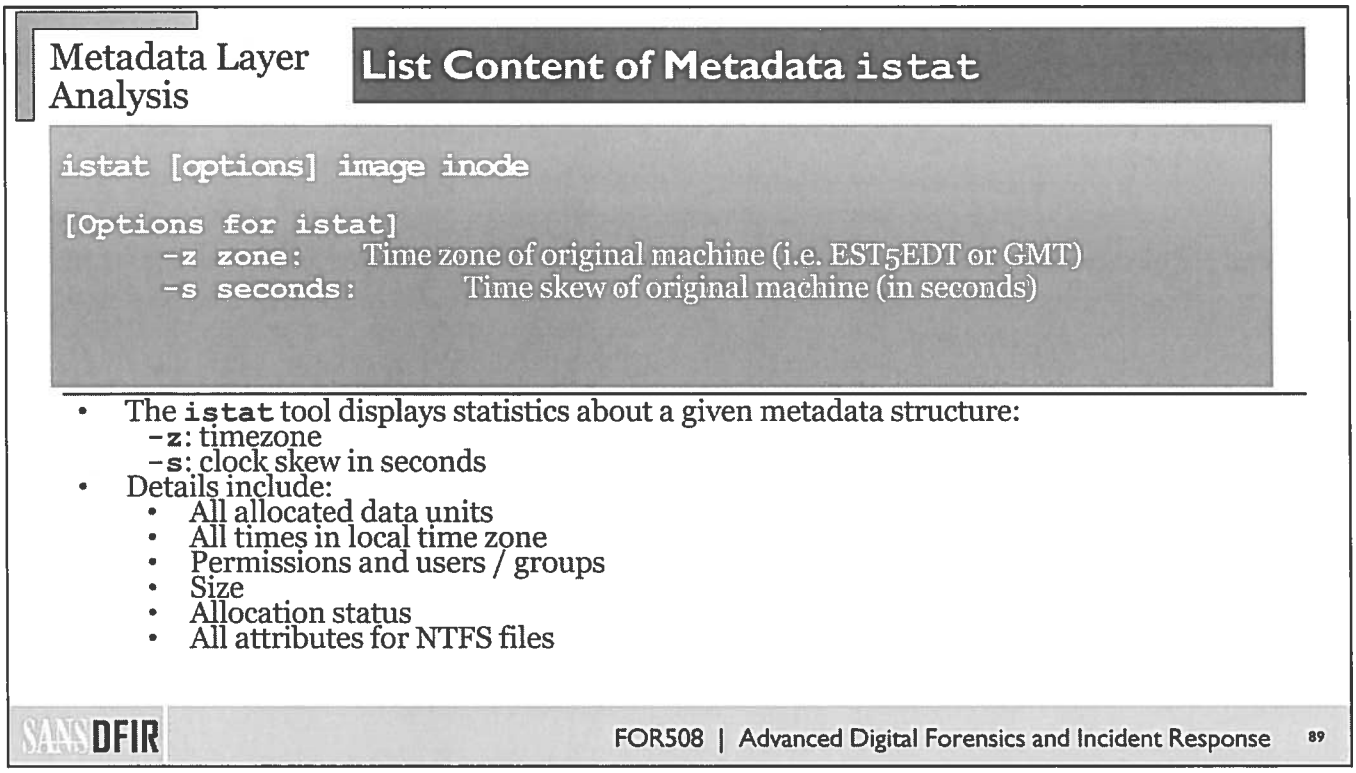
-r:	Recover deleted file
-R:	Recover deleted file and suppress recovery errors
-s:	Display slack space at end of file

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 2,000 bytes in size and has a 4,096 byte cluster allocated to it. Normally, 'icat' will output only the 2,000 bytes, but with '-s', then 'icat' will output the full 4,096 bytes from the cluster.

The '-r' flag causes 'icat' to perform file recovery for some filesystems, specifically with FAT. Some filesystems, 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:

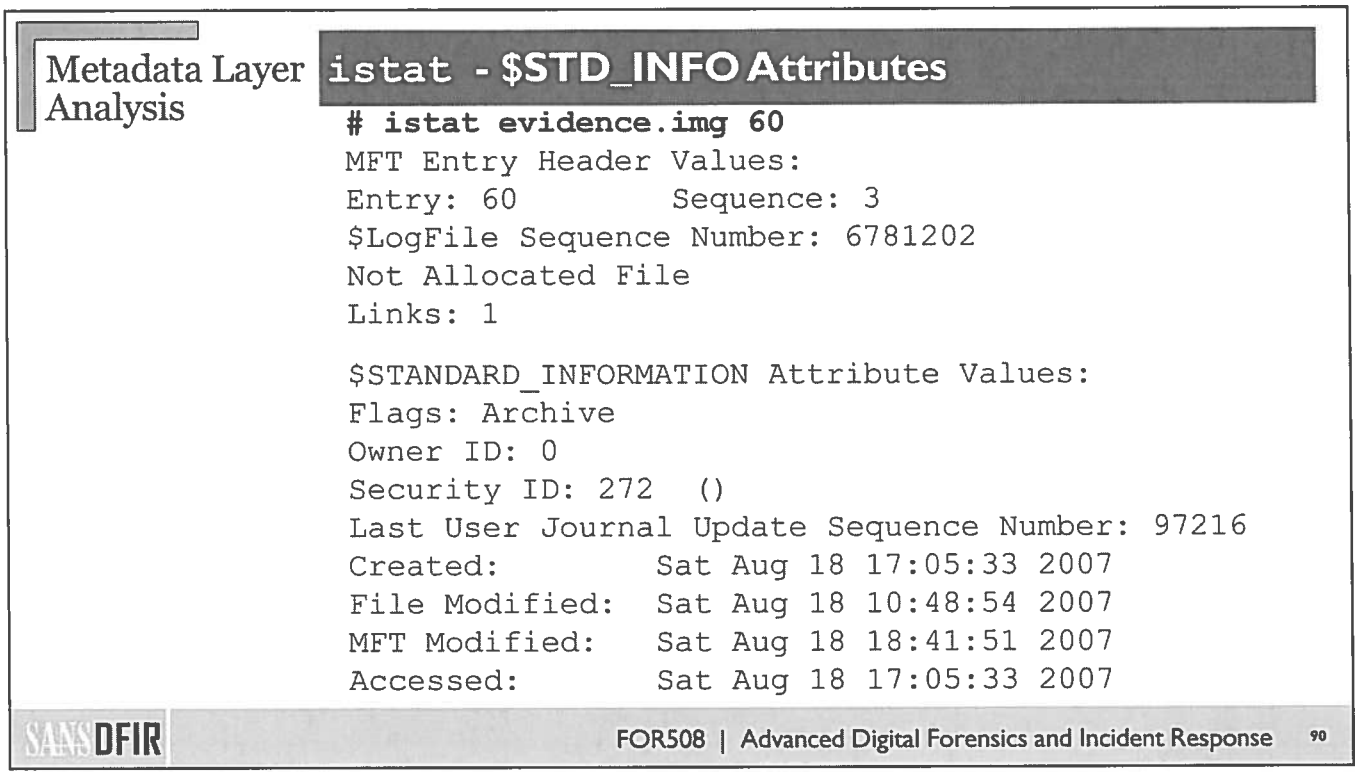
```
istat [options] image inode

[Options for istat]
        -z zone:    Time zone of original machine (that is, 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 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 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 were slow by 32 seconds, then the flag '-s -32' would be given. Then, both the adjusted and un-adjusted times would be displayed.



Here we see the output from running 'istat' on an NTFS file. NTFS is a more complex filesystem 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.

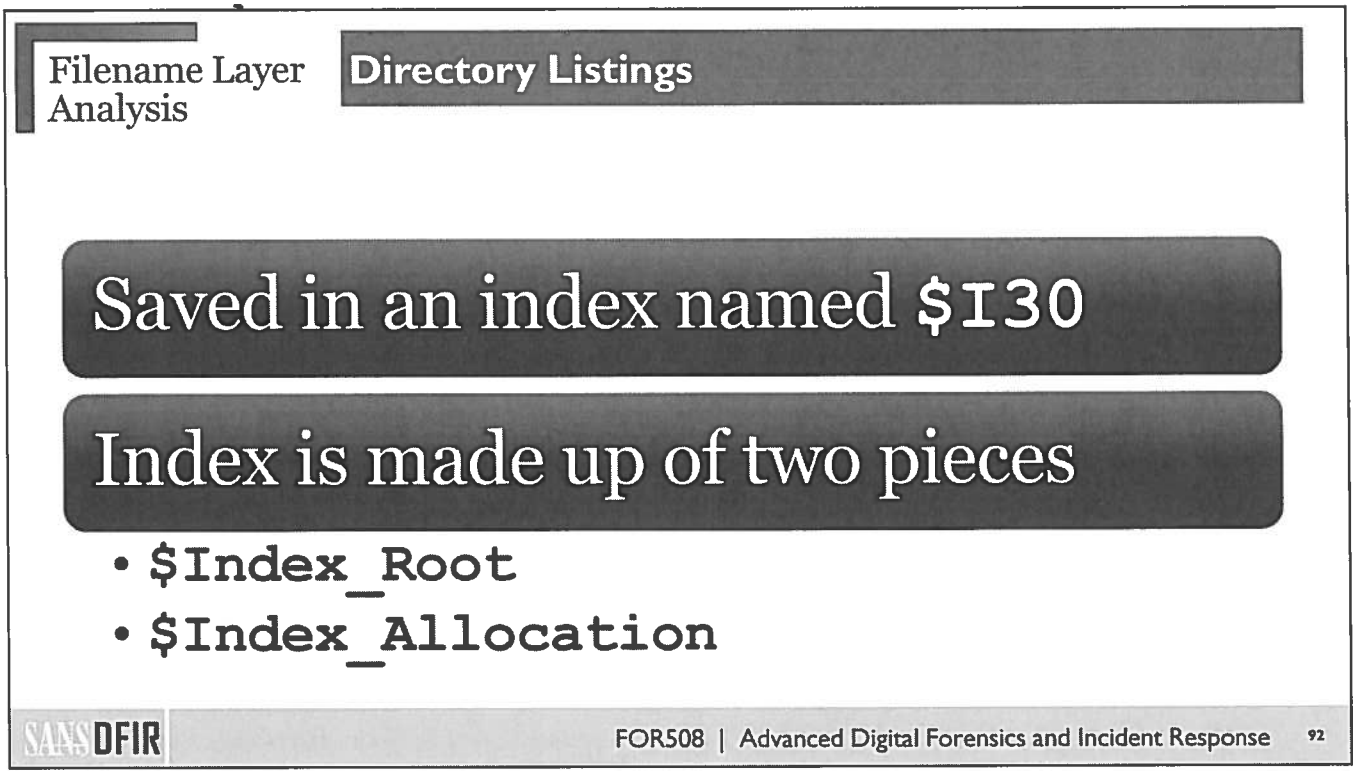
Metadata Layer Analysis

istat - $FILE_NAME Attributes

```
$FILE_NAME Attribute Values:
Flags: Archive
Name: PS.doc
Parent MFT Entry: 46    Sequence: 1
Allocated Size: 24576          Actual Size: 22016
Created:                Sat Aug 18 17:05:33 2007
File Modified:  Sat Aug 18 10:48:54 2007
MFT Modified:   Sat Aug 18 17:05:34 2007
Accessed:               Sat Aug 18 17:05:33 2007

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 72
Type: $FILE_NAME (48-3)   Name: N/A   Resident   size: 78
Type: $OBJECT_ID (64-4)   Name: N/A   Resident   size: 16
Type: $DATA (128-1)   Name: $Data   Non-Resident   size: 22016
186082 186083 186084 186085 186086 186087
```


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 the attributes are given. The $DATA attribute is where the file content is stored, and we see that it is 22,016 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 timestamps 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 filename 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 filename 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 filesystems that do not delete the pointer from the filename structure to the metadata structure, the deleted filename can be found.

Filename Layer Analysis

$130

Index Header

Index Entry

Index Entry

x00	49	4E	44	58	28	00	09	00	66	9F	7B	6C	0B	00	00	00	INDX(...f?{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..... .....
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...
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!.?bØË.
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!.?bØË.. .....
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.

SANS DFIR FOR508 | Advanced Digital Forensics and Incident Response 94

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 4,096 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 have only a couple of files in a directory, their entries will take up only 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 timestamps, size, flags, and the filename. 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. When 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

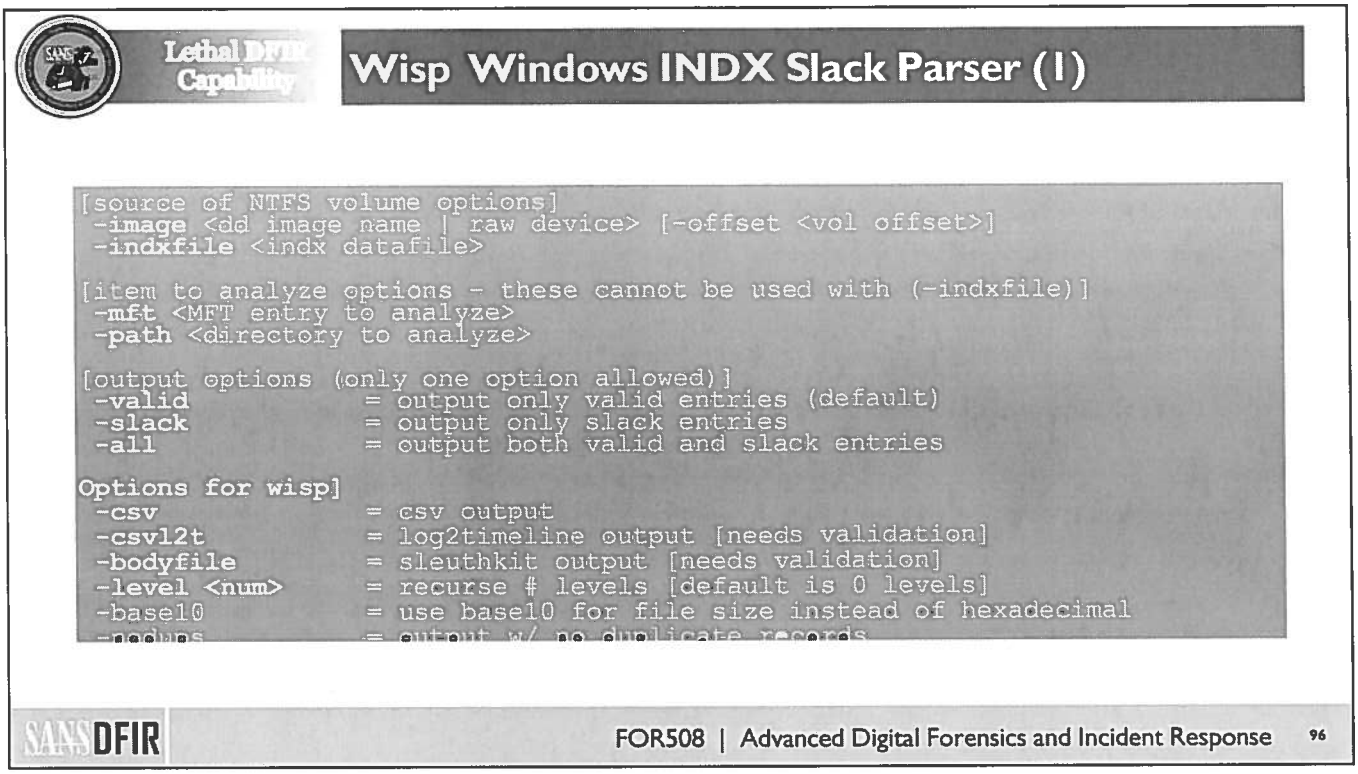
```
x00  49 4E 44 58 28 00 09 00 66 9F 7B 6C 0B 00 00 00  INDX(...fŸ{l....
     INDX Signature
x10  00 00 00 00 00 00 00 00 28 00 00 00 70 07 00 00  ........(...p...
                                         Size of Entries
X20  E8 0F 00 00 00 00 00 00 64 0B 01 00 07 00 74 00  è.......d.....t.
     Allocated 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!.ŸbØË.
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!.ŸbØË.. ......
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 versus 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.

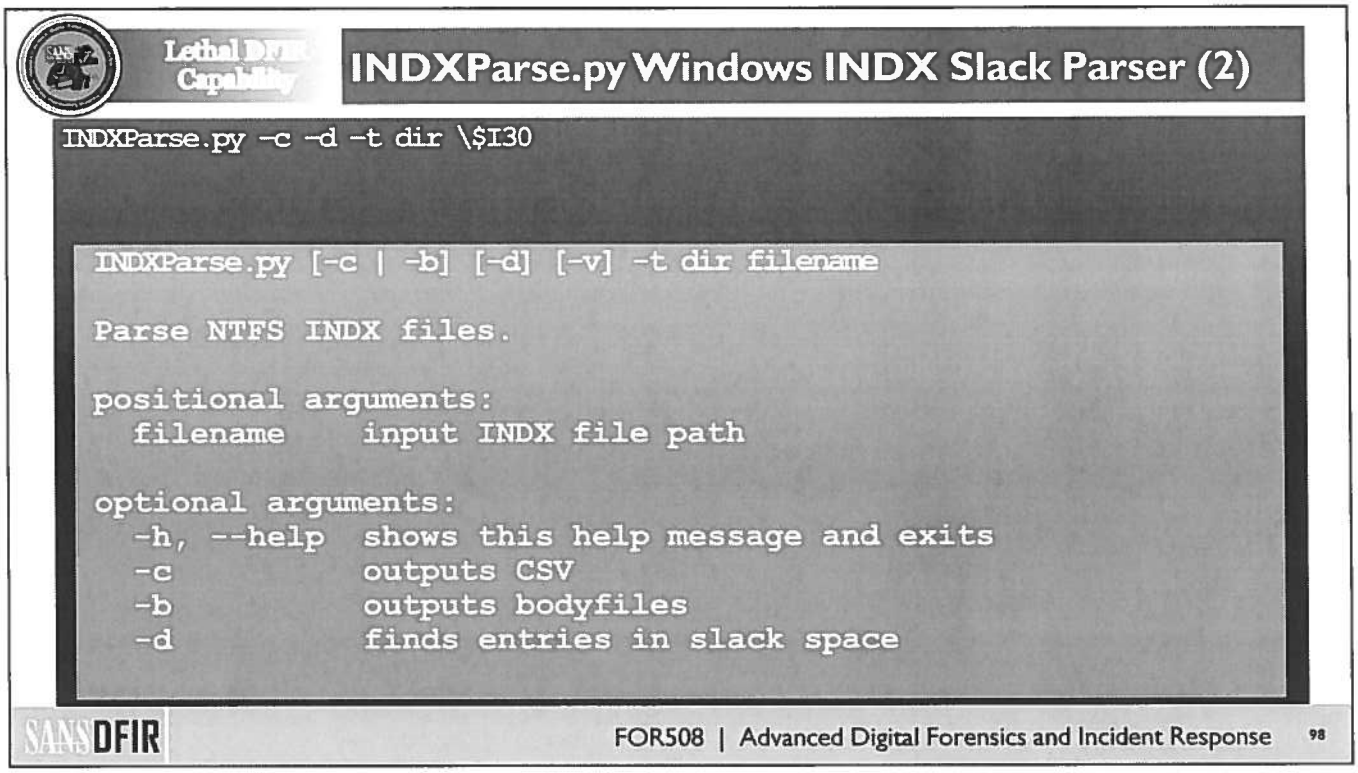
As a directory's contents are changed, the number of valid index entries grows or shrinks, as appropriate. As more directory entries are added, eventually it will exceed the existing INDX record allocation space. At this point the operating system will allocate an additional INDX record in the size of 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]

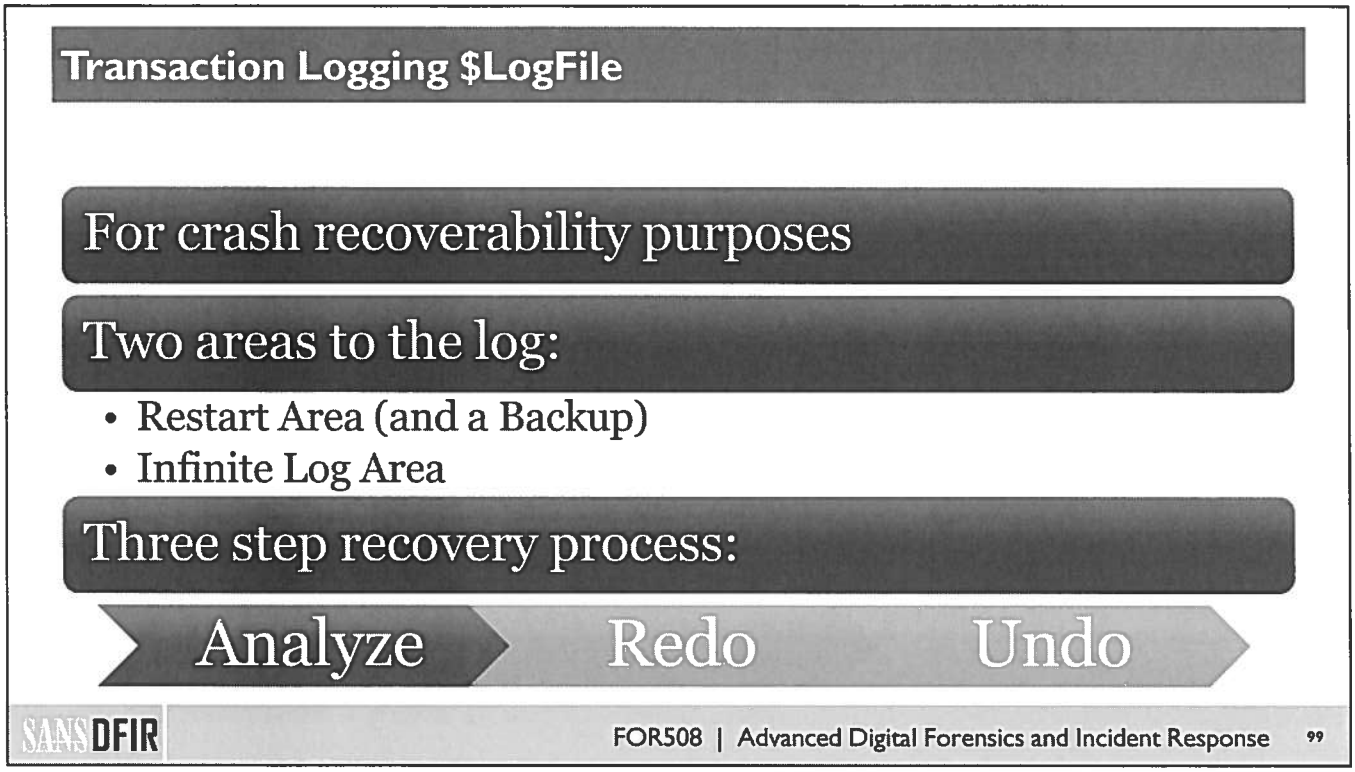
References
[1] - http://www.williballenthin.com/forensics/indx/index.html
[2] - http://writeblocked.org/dfironline.html

```
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
```

```
root@siftworkstation:/cases# INDXParse.py -c -d -t dir \$I30
FILENAME,       PHYSICAL SIZE, LOGICAL SIZE,   MODIFIED TIME,  ACCESSED TIME,  CHANGED TIME,   CREATED TIME
144A~1, 64,     64,     2012-04-06 18:15:03.072256,     2012-04-06 18:15:01.728497,     2012-04-06 18:15:59.260115,     2012-04-06 18:15:01.728497
system4,        0,      0,      2012-04-06 18:20:12.354105,     2012-04-06 18:20:12.354105,     2012-04-06 18:20:12.354105,     2012-04-05 14:36:25.928574
system4.rar,    6299648,        6297428,        2012-04-06 13:53:09.349937,     2012-04-06 13:52:56.321257,     2012-04-06 13:53:09.349937,     2012-04-06 13:48:28.908258
system5,        0,      0,      2012-04-06 18:20:24.837984,     2012-04-06 18:20:24.837984,     2012-04-06 18:20:24.837984,     2012-04-05 14:28:31.368065
system6,        0,      0,      2012-04-06 18:20:35.737816,     2012-04-06 18:20:35.737816,     2012-04-06 18:20:35.737816,     2012-04-05 17:55:44.132687
system7,        0,      0,      2012-04-06 18:20:45.825117,     2012-04-06 18:20:45.825117,     2012-04-06 18:20:45.825117,     2012-04-05 14:15:13.603449
0ç¬0¶, 64,      64,     2012-04-06 18:15:03.072256,     2012-04-06 18:15:01.728497,     2012-04-06 18:15:59.260115,     2012-04-06 18:15:01.728497
0ç¬0¶ (slack at 0x2e8),         64,     64,     2012-04-06 18:15:03.072256,     2012-04-06 18:15:01.728497,     2012-04-06 18:15:59.260115,     2012-04-06 18:15:01.728497
0ç¬0¶ (slack at 0x348),         64,     64,     2012-04-06 18:15:03.072256,     2012-04-06 18:15:01.728497,     2012-04-06 18:15:59.260115,     2012-04-06 18:15:01.728497
```

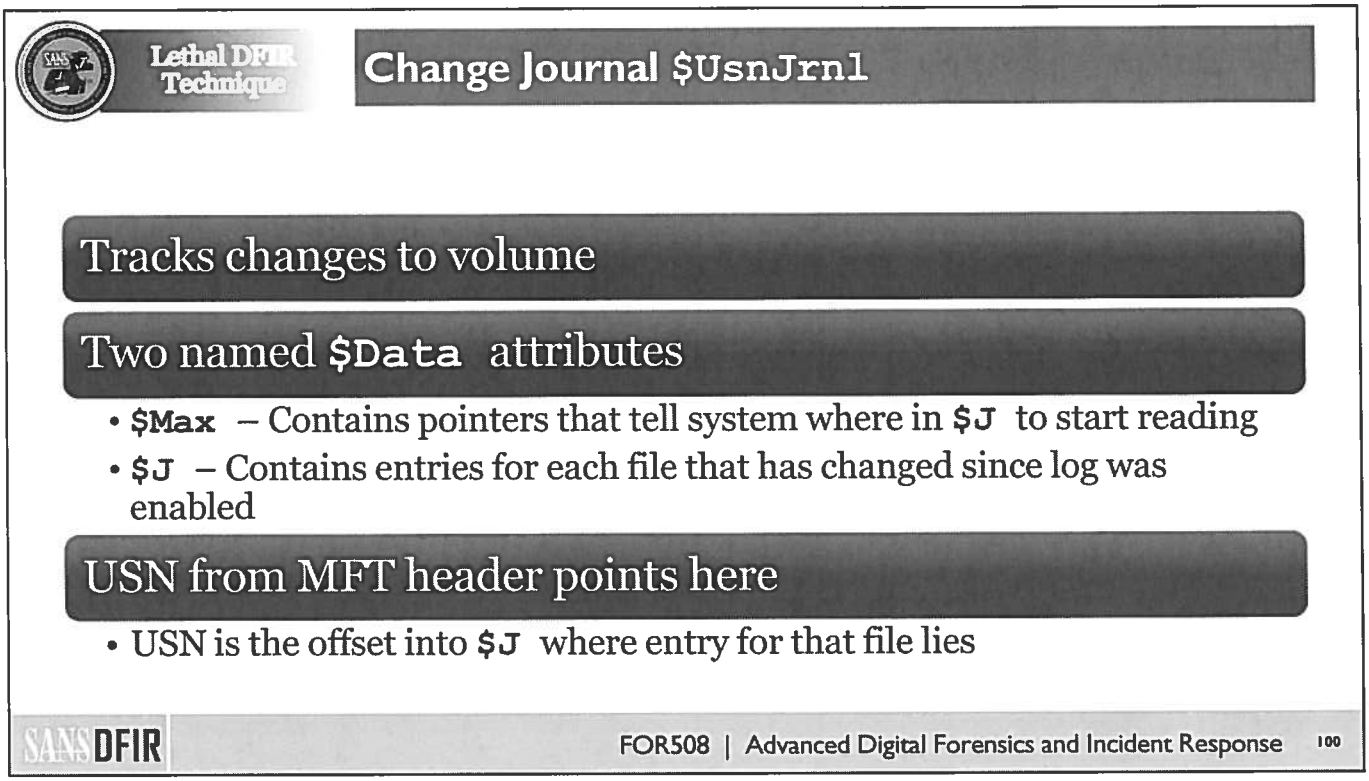


NTFS uses the $LogFile to maintain a log of all file transactions, such as a read, write, or modification of any file. It takes multiple steps to update multiple structures to write to a single file; the $logFile keeps track of each of those steps so that the state of each transaction can be tracked. When 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 have to recover only from the 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 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 the 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 filename of $USNJrnl), but all 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 useful for utilities that need to do incremental scans of a system, such as backup or antivirus 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.

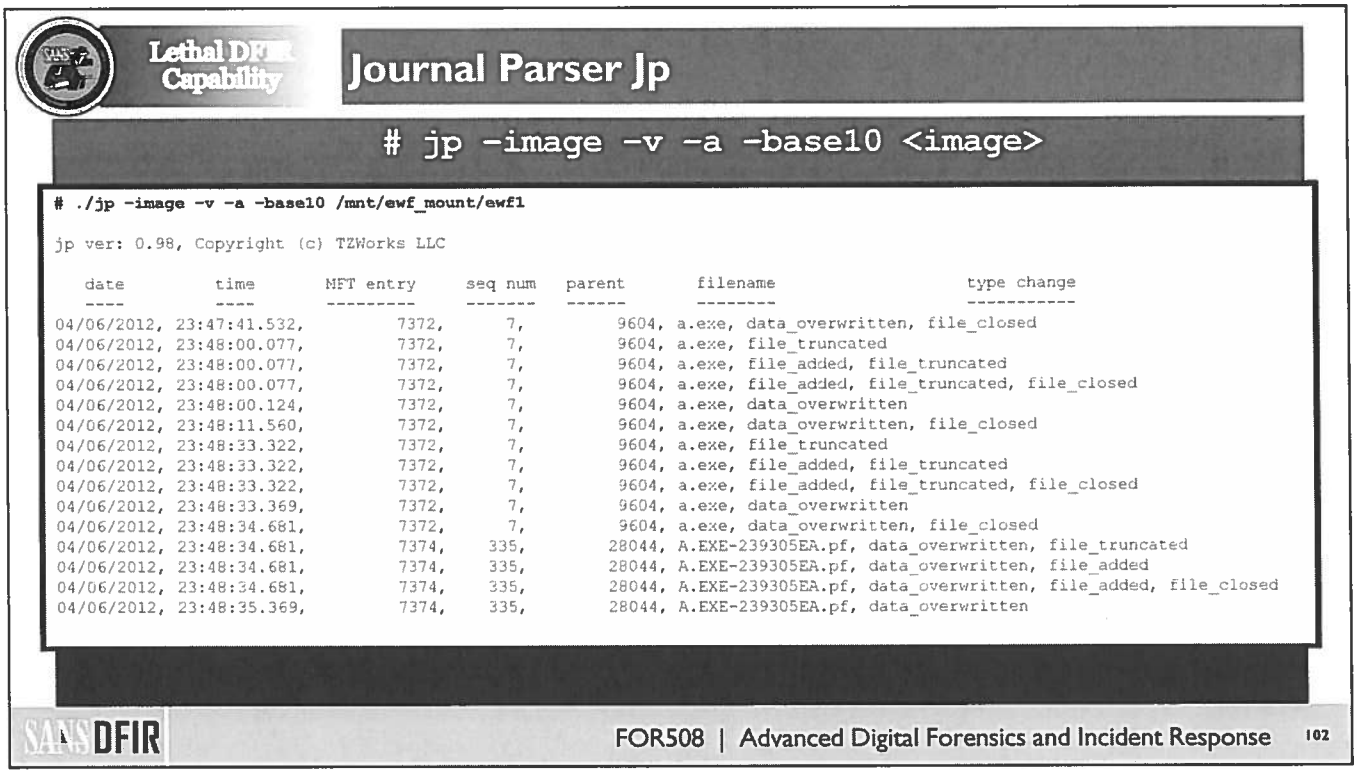
0x00 4 Size of entry
0x04 2 Major Version
0x06 2 Minor Version
0x08 8 MFT Reference
0x10 8 Parent MFT Reference
0x18 8 Offset of this entry in $J
0x20 8 Timestamp
0x28 4 Reason (see below)
0x2B 4 SourceInfo
0x30 4 SecurityID
0x34 4 FileAttributes
0x38 2 Size of filename (in bytes)
0x3A 2 Offset to filename
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, filename changed, attributes or times changed, compression state changed, encryption state changed, and 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 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



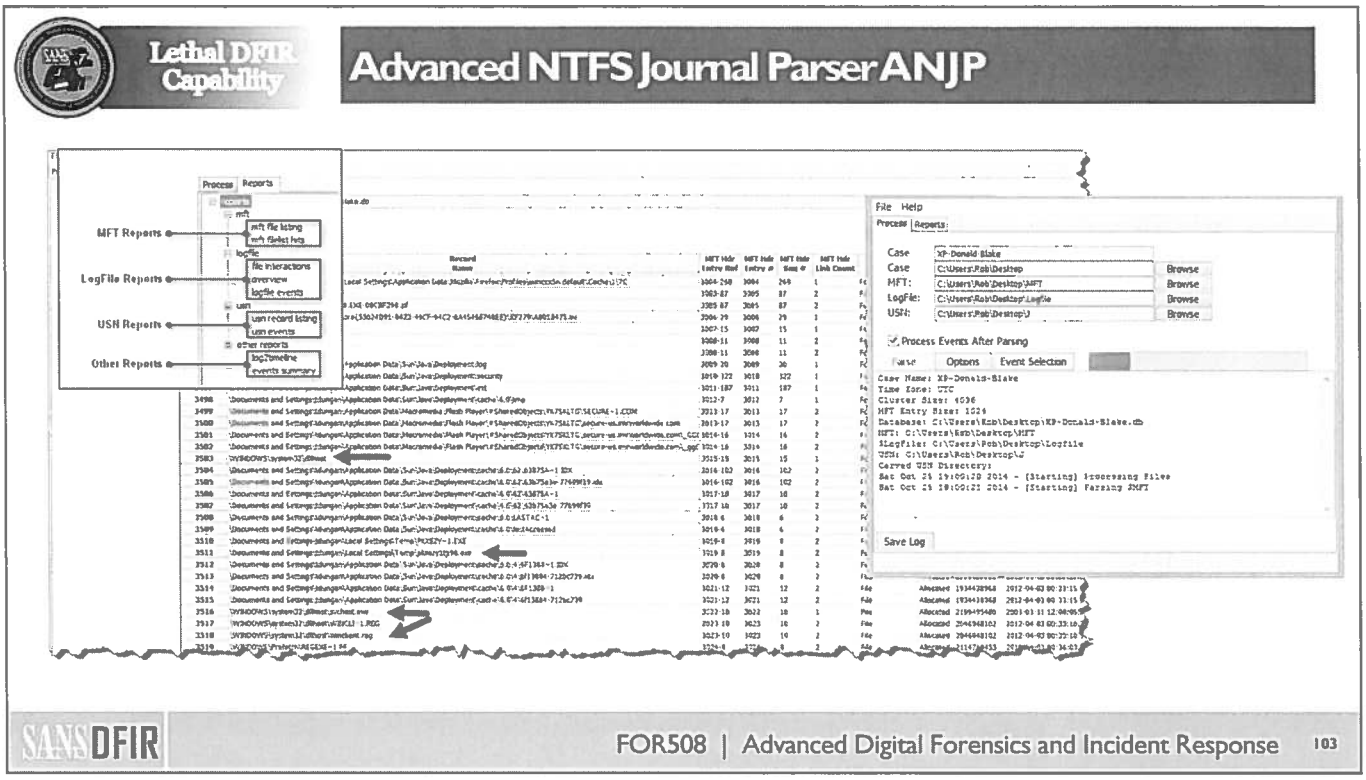
JP[1] is a small command-line tool that does an outstanding job of parsing out the $UsnJrnl:$J index. This tool is cross-platform, so there is a version for both the x86 and 64-bit versions of Windows, Linux, and Mac. It can parse the journal from a mounted filesystem, a raw disk image, or an extracted $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 #, filename, and changes. 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 filename, 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 filesystem: MFT, LogFile, and USN.

ANJP allows examiners to view filesystem 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 filesystem 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 important, 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 filesystem 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 MFT recorded 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.

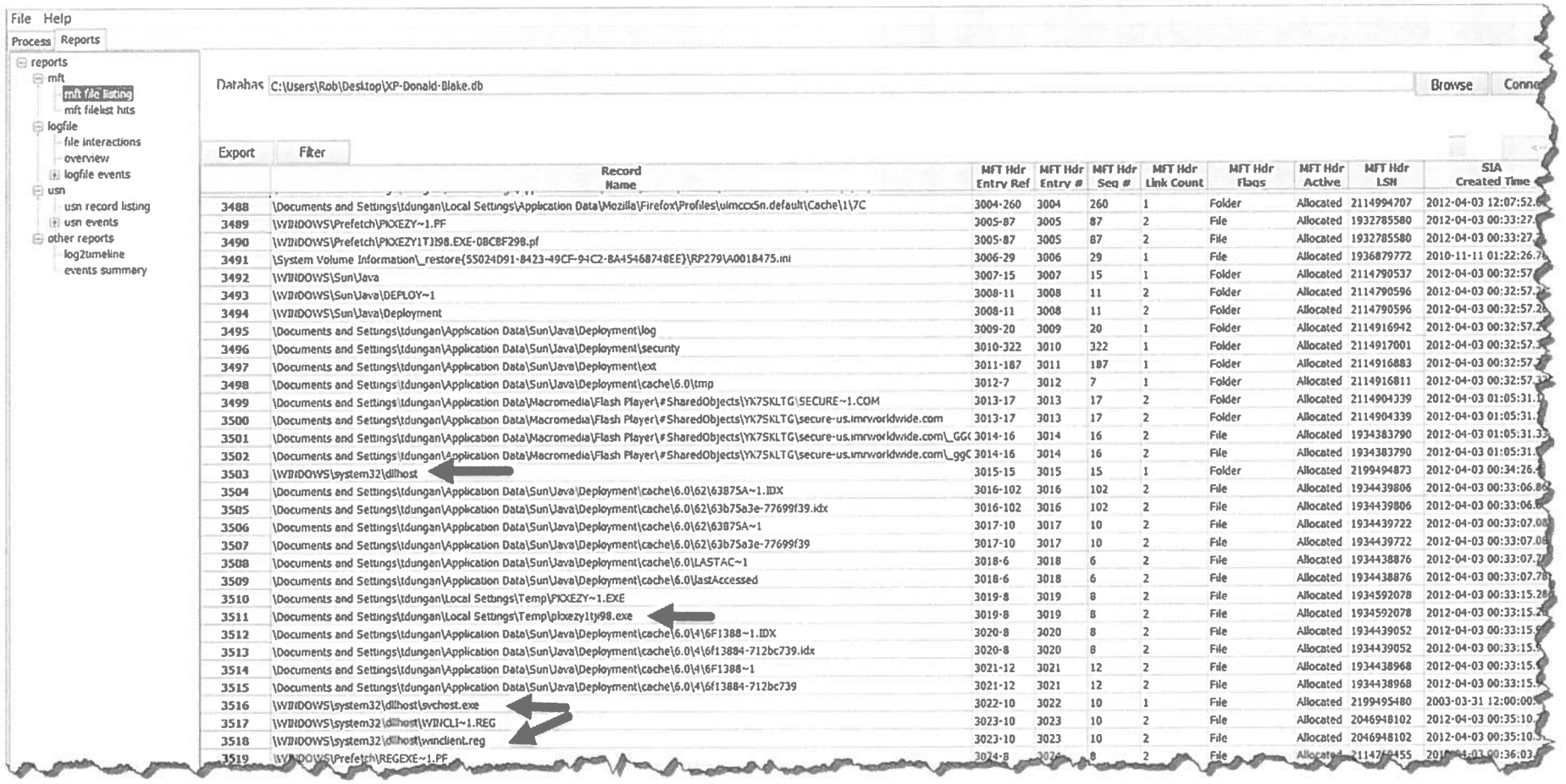
By linking the LogFile and USN to the MFT, the fullpath (path and filename) for a given record within a LogFile or USN record can be enumerated. However, this linkage cannot be used to maintain fullpaths while parsing the entire LogFile or USN Journal. This is because as records are added to the LogFile, 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.

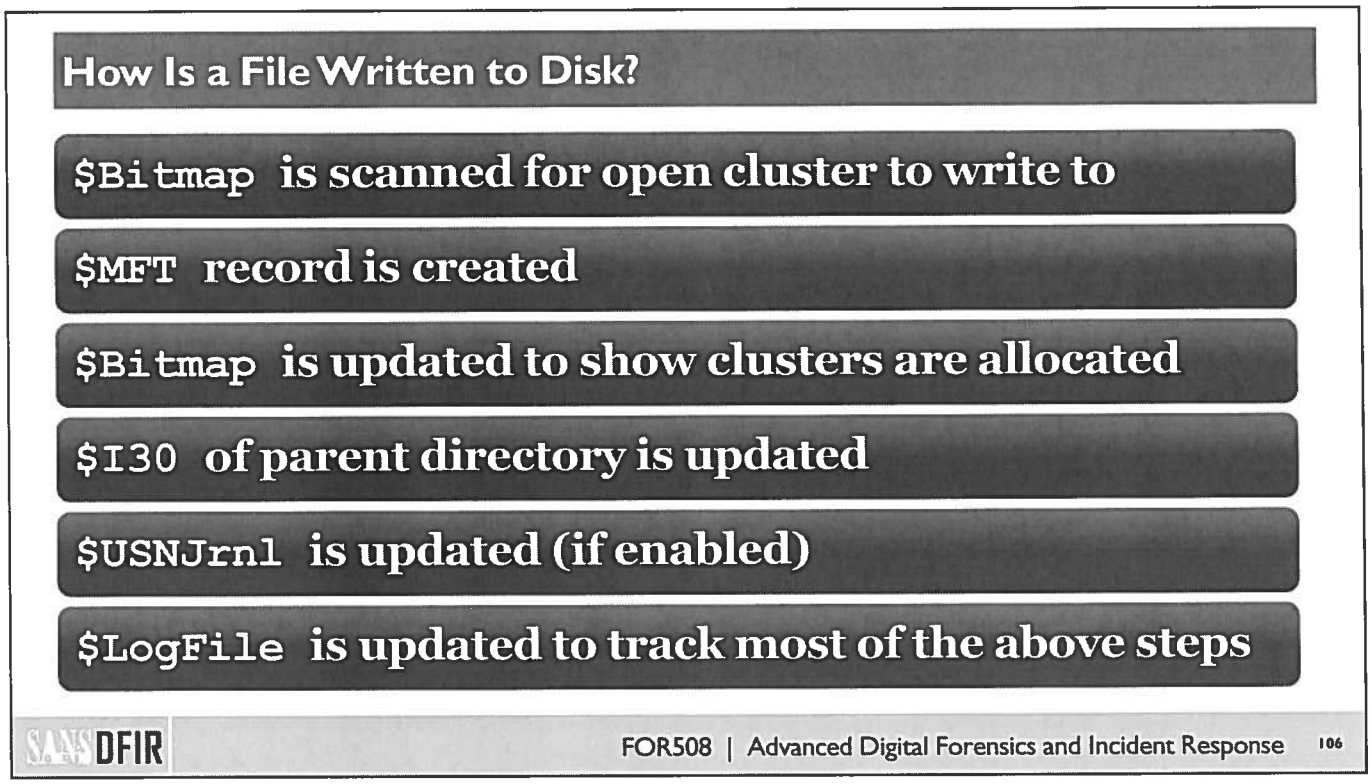
Thus, parsing the LogFile and USN records from newest to oldest records, and applying changes to the fullpaths as files are deleted, created, and renamed results in knowing exactly where a file was located and what its name was, when a change occurred.

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.

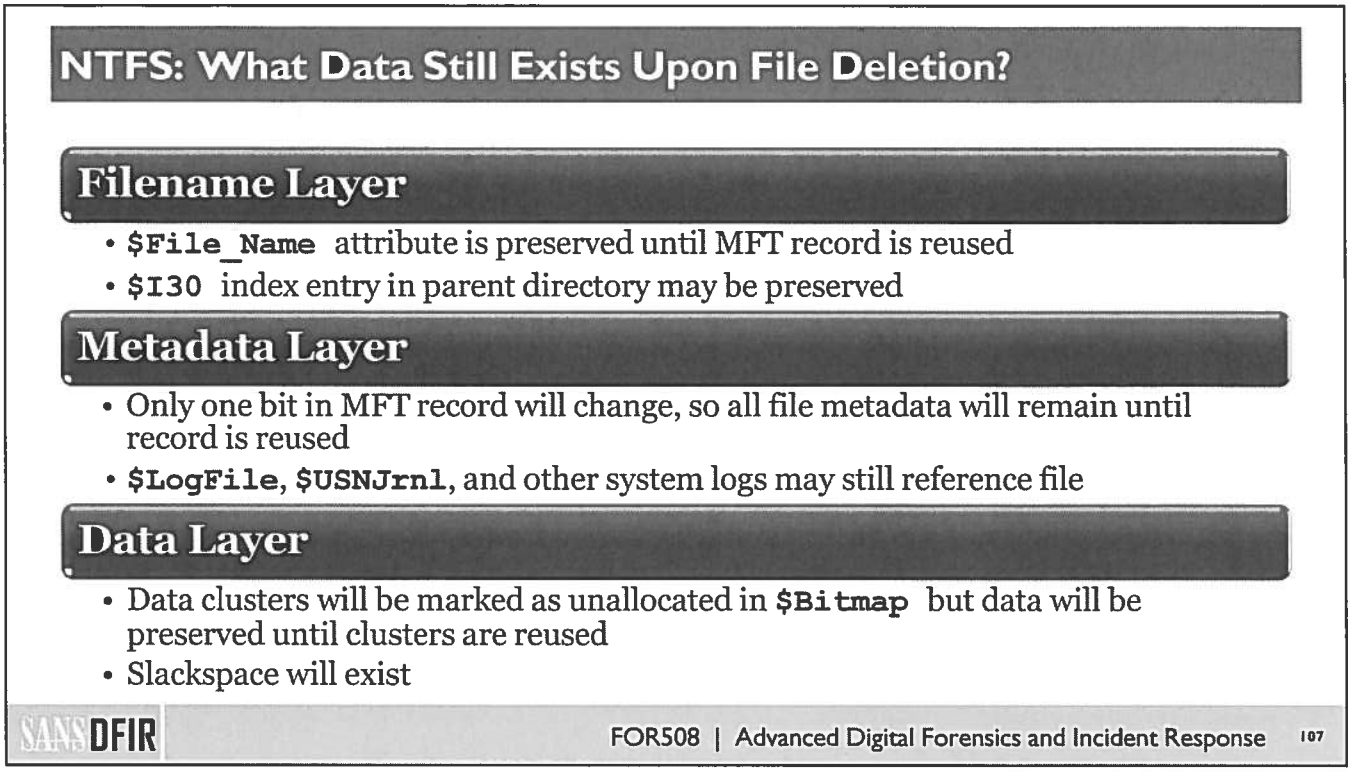
[1] https://www.gettriforce.com/product/anjp-free/


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
Database C:\Users\Rob\Desktop\XP-Donald-Blake.db
Browse
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\uimccc5n.default\Cache\1\7C 3004-260 3004 260 1 Folder Allocated 2114994707 2012-04-03 12:07:52.6
3489 \WINDOWS\Prefetch\PIXEZY~1.PF 3005-87 3005 87 2 File Allocated 1932785580 2012-04-03 00:33:27.
3490 \WINDOWS\Prefetch\PIXEZY1TJ98.EXE-0BCBF298.pf 3005-87 3005 87 2 File Allocated 1932785580 2012-04-03 00:33:27.
3491 \System Volume Information_restore{55024D91-8423-49CF-94C2-BA45468748EE}\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 00:32:57.26
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 00:32:57.3
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 01:05:31.0
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 01:05:31.33
3502 \Documents and Settings\tdungan\Application Data\Macromedia\Flash Player\#SharedObjects\YK7SKLTG\secure-us.imrworldwide.com_ggC 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\63875A~1.IDX 3016-102 3016 102 2 File Allocated 1934439806 2012-04-03 00:33:06.86
3505 \Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\62\63b75a3e-77699f39.idx 3016-102 3016 102 2 File Allocated 1934439806 2012-04-03 00:33:06.8
3506 \Documents and Settings\tdungan\Application Data\Sun\Java\Deployment\cache\6.0\62\63875A~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 1934438876 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 1934438876 2012-04-03 00:33:07.78
3510 \Documents and Settings\tdungan\Local Settings\Temp\PIXEZY~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\pixezy1tj98.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 12:00:00
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\wnclient.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 2114769455 2012-04-03 00:36:03.



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

To improve performance, all these writes will take place in memory in a cache maintained by the filesystem 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, for example, 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 writes to the volume coming in at the same time, a queue could build up and could take a little while before it is all committed to disk.



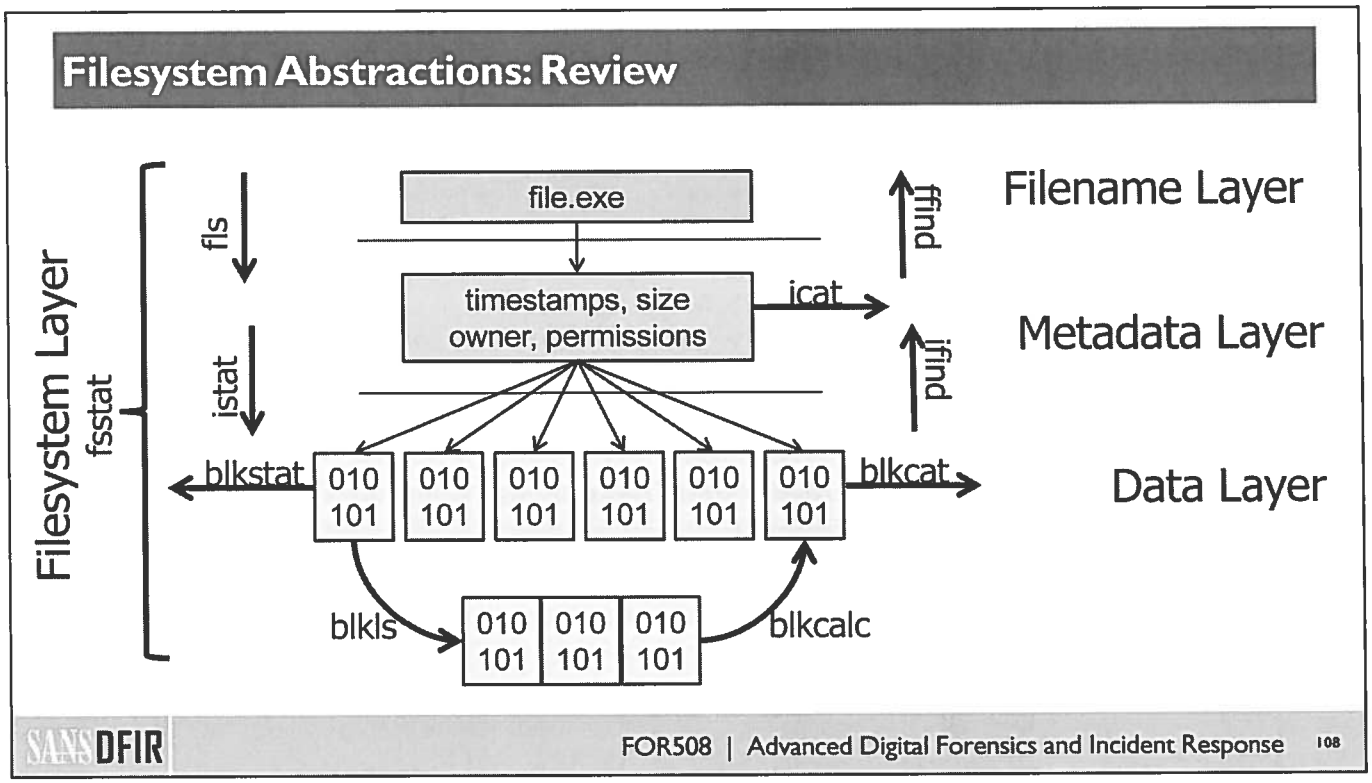
When a file is deleted, most of the steps from the previous slide are reversed.

- 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).

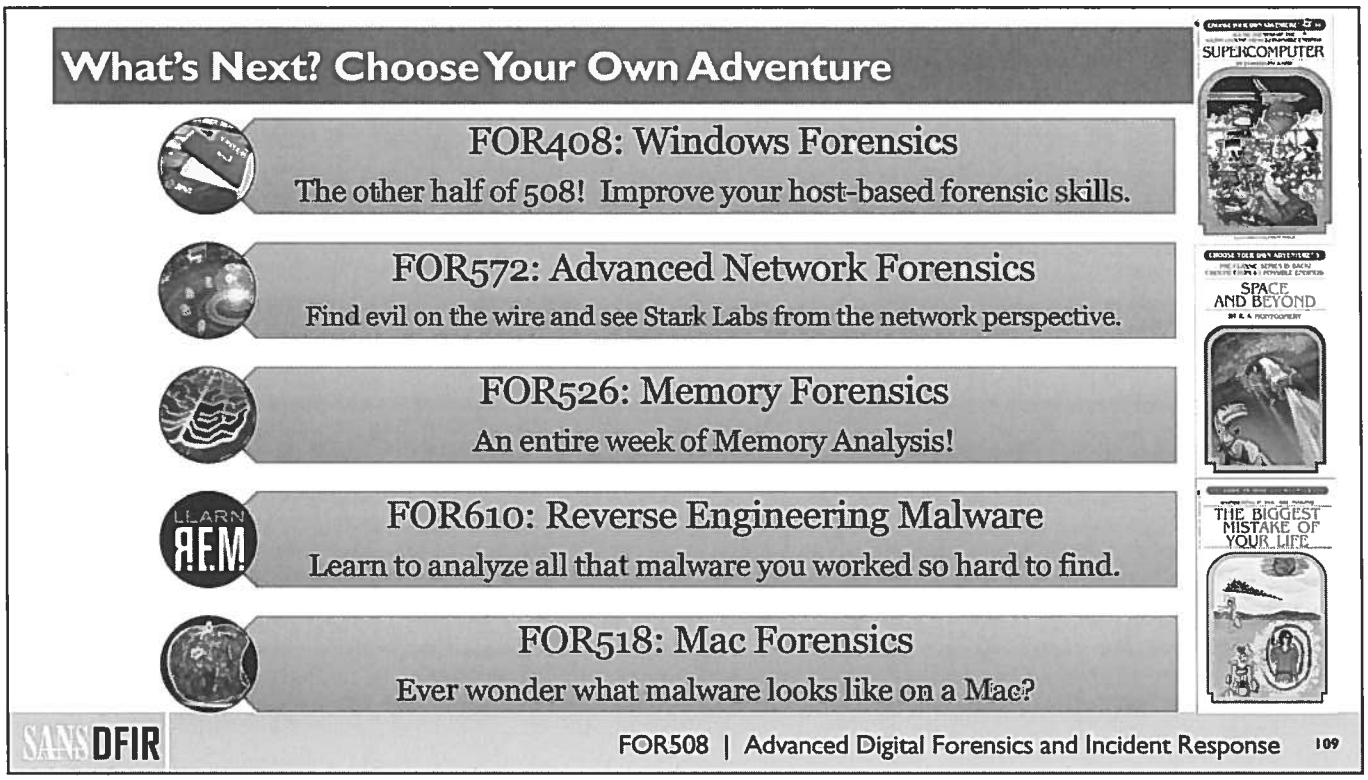
Just as with FAT and other filesystems, deleting a file deallocates the clusters by updating the $Bitmap file but leaves the data in place for us forensics examiners to look at later. The MFT Record has the "in use" flag (a single bit) flipped in the Record header to mark that the record is no longer in use. MFT Records will get reused, causing the deleted file's metadata to be overwritten. Sometimes this happens right away, whereas 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 filesystem abstraction categories and how various Sleuthkit Tools graphically relate to each other.

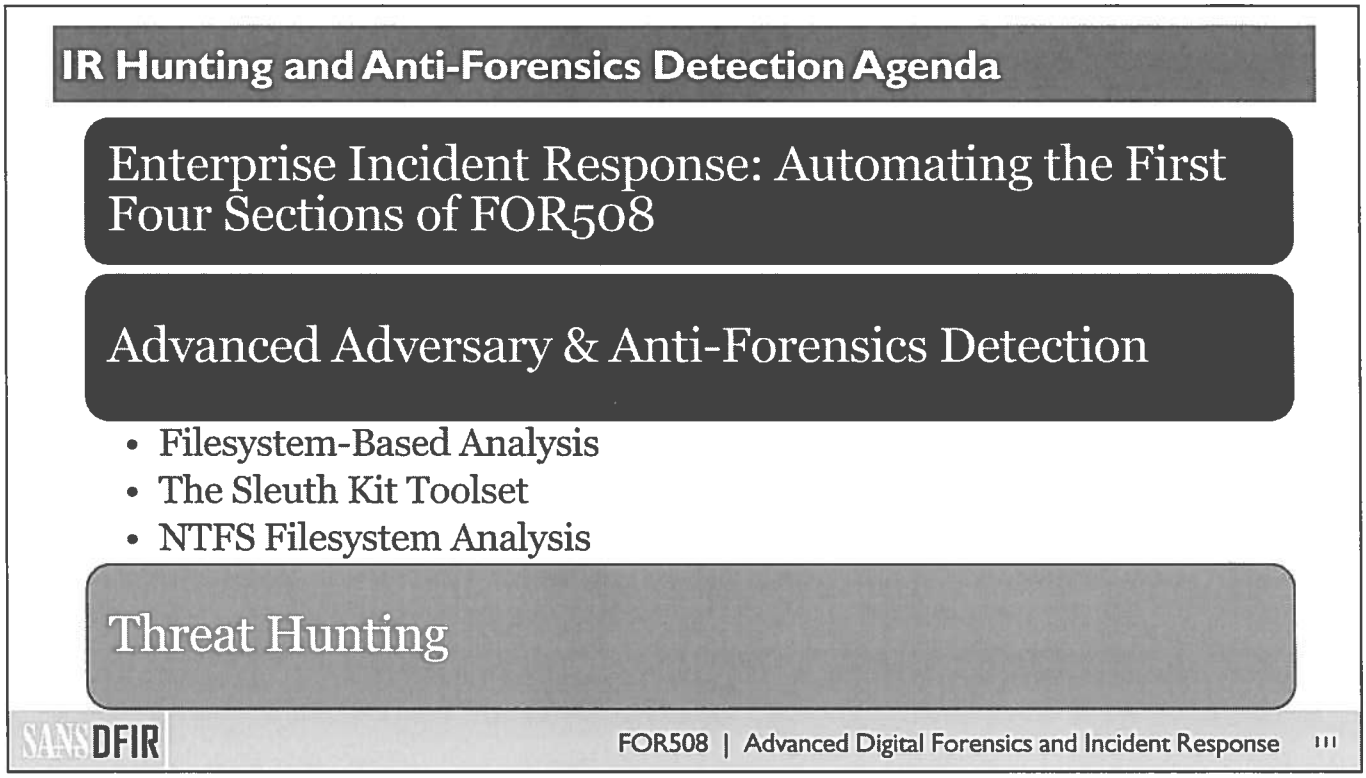


This page intentionally left blank.

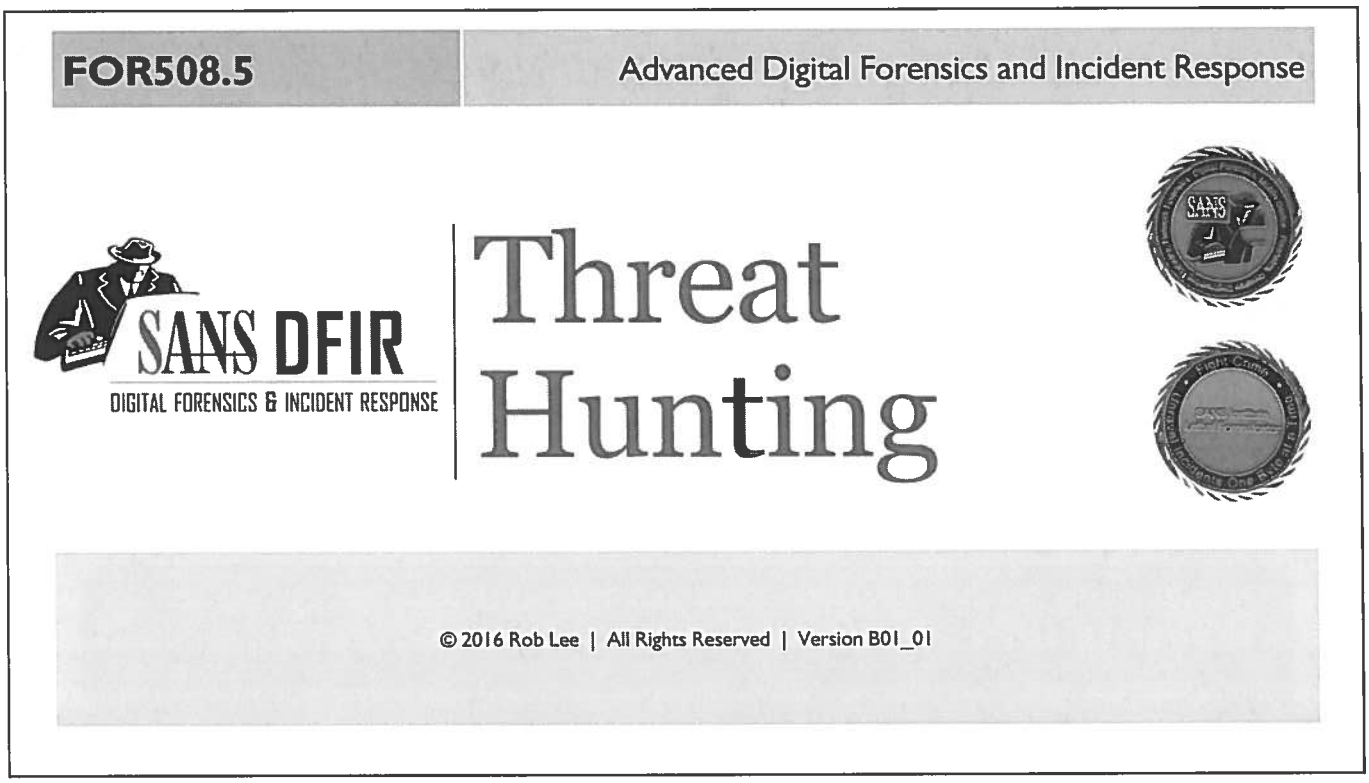
Exercise 5.2

Anti-Forensics Detection and NTFS

This page intentionally left blank.



This page intentionally left blank.



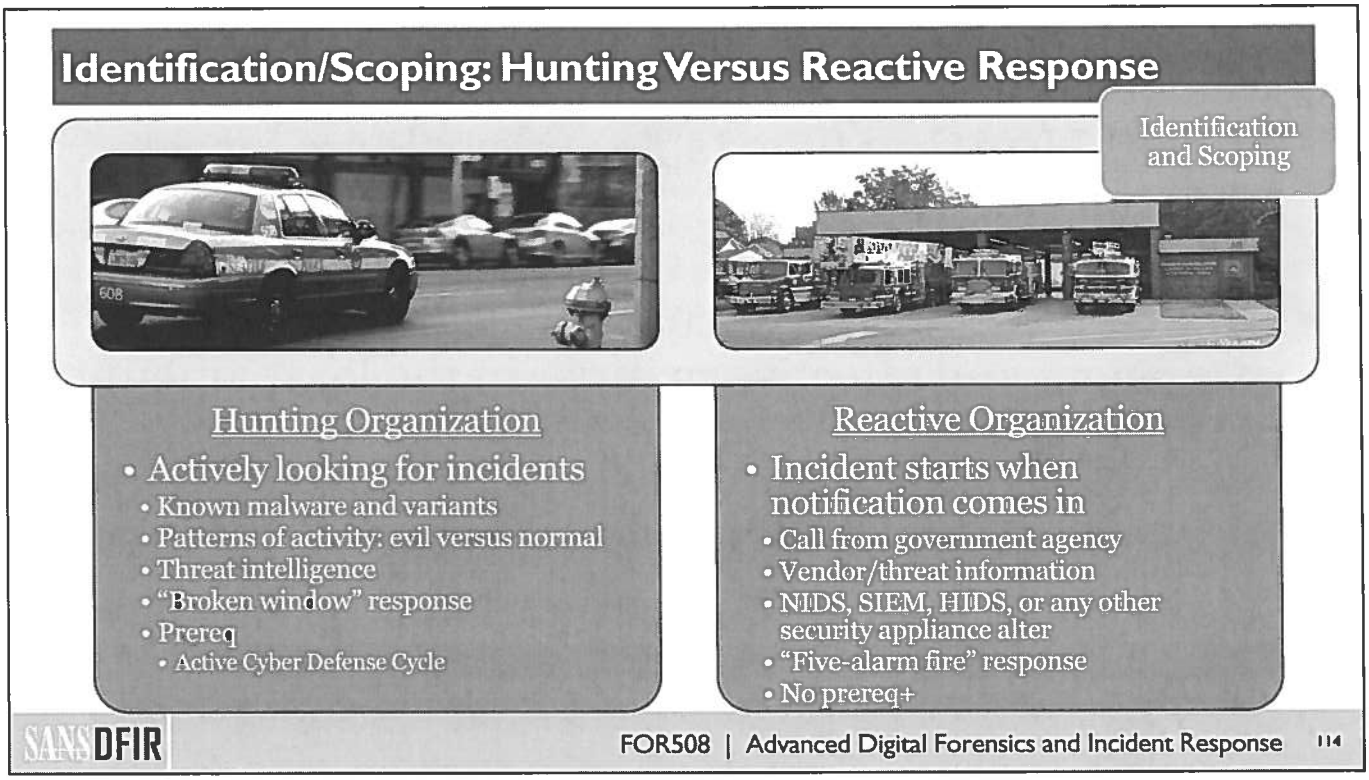
Author: Rob Lee
rlee@sans.org
http://twitter.com/robtlee
http://twitter.com/sansforensics

Threat Hunting vs. Incident Response

- **Threat Hunting** time table and goals are longer term
- **Incident Response** focuses on putting out "fire" in efficient and effective manner in the least amount of time
- Result: Tactics change when you Hunt

Especially if you have more TIME.

This page intentionally left blank.



Identification/Scoping: Hunting Versus Reactive Response

Hunting is about taking a proactive versus a reactive approach to identifying incidents.

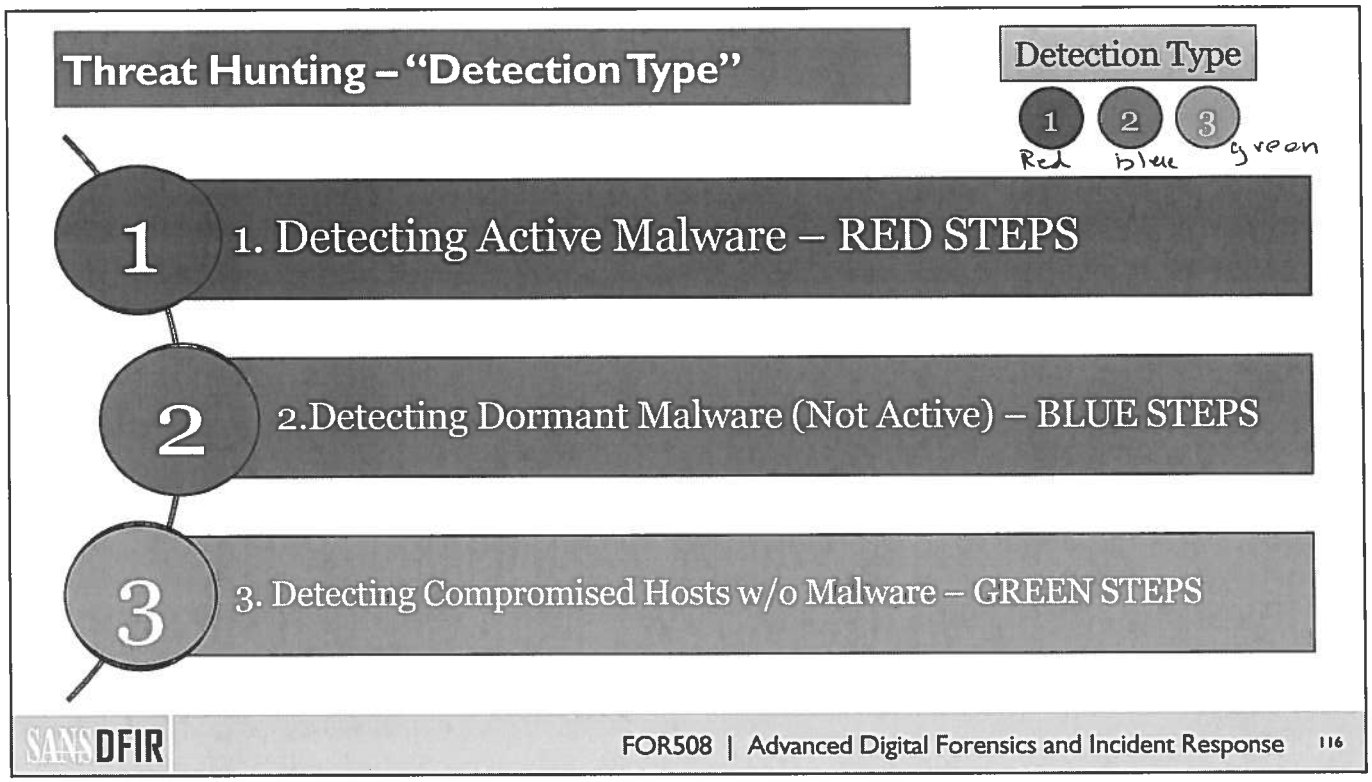
A reactive organization begins incident response when an alert or notification comes in. The alert could come from a third party such as the FBI, or it could come from the organization's own security sensors. The best analogy to a reactive approach is that the IR team is largely waiting to be called into action and relying on accuracy of the notifications it is receiving. Most organizations start building their incident response teams as a reactive organization and there is nothing wrong with that. In many cases, the IR team is largely comprised of augmentation staff that normally fulfill other duties during their regular jobs. As the organization grows larger or if it has an increasing number of incidents, the team is likely to become permanent. The best analogy here is small towns with "volunteer fire departments" vs. "full-time fire fighters." Even larger organizations likely still augment their IR teams with additional personnel internally or might even contract to third-party contractors who provide incident response services.

Organizations move from a reactive organization to a hunting organization when they realize they are not detecting their incidents early enough. The idea of hunting-based response doesn't mean it is an "either or" approach. Most hunting organizations are also reactive organizations, but they begin to task their incident response team to actively engage and hunt for adversaries inside their environment. To accomplish this task, the hunting team typically will be armed with known malware, patterns of activity, or accurate threat group intelligence aiding them in their search.

Organizations who decide to create a hunting organization sometimes fail to see the importance of proper threat intelligence for driving the search in the right areas. Simply tasking a team to "find evil" isn't enough. The team needs to know the difference between normal and abnormal. It needs to know typical hacker tools and techniques. It needs to be skilled in both network and host-based forensics and response to look for the

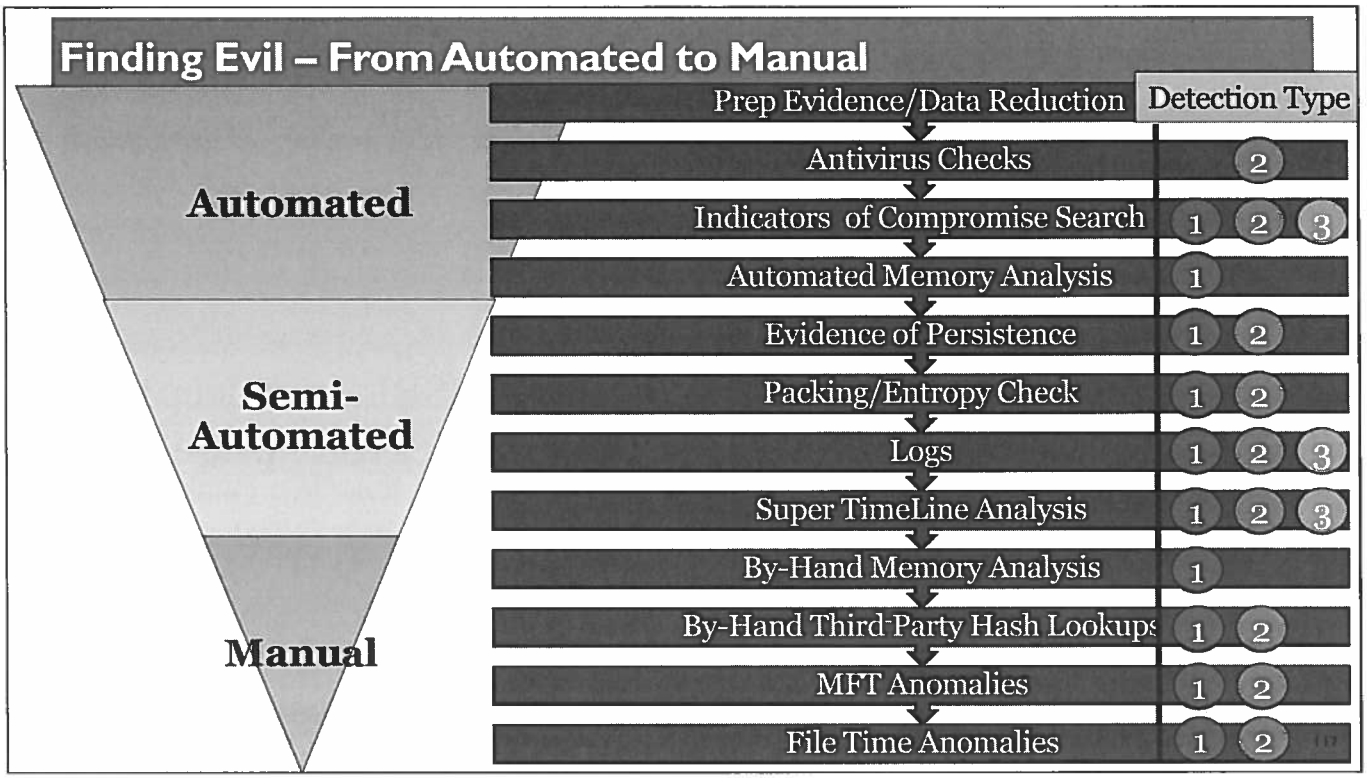
footprints of these adversaries. Finally, it helps if the organization has invested heavily into a cyber threat intelligence capability that will accurately help guide the team to the right locations on the network to look for specific indicators associated with threat groups interested in that particular data or capability the organization owns.

Without any type of threat intelligence even in its basic form of patterns of typical hacker activity, most hunting groups are simply tasked with looking for "things that look weird" without knowing what weird versus normal even looks like. A trained hunter must know the difference between normal and abnormal as a prerequisite. Even better, if a threat intelligence capability is informing the team, it would look for specific threat groups targeting specific programs using specific techniques. This is an achievable goal. Hunting involves both a manual and automated scanning of systems looking for evil. I have seen some organizations use only network data and do not have proper system/host interrogation capabilities paired with it.

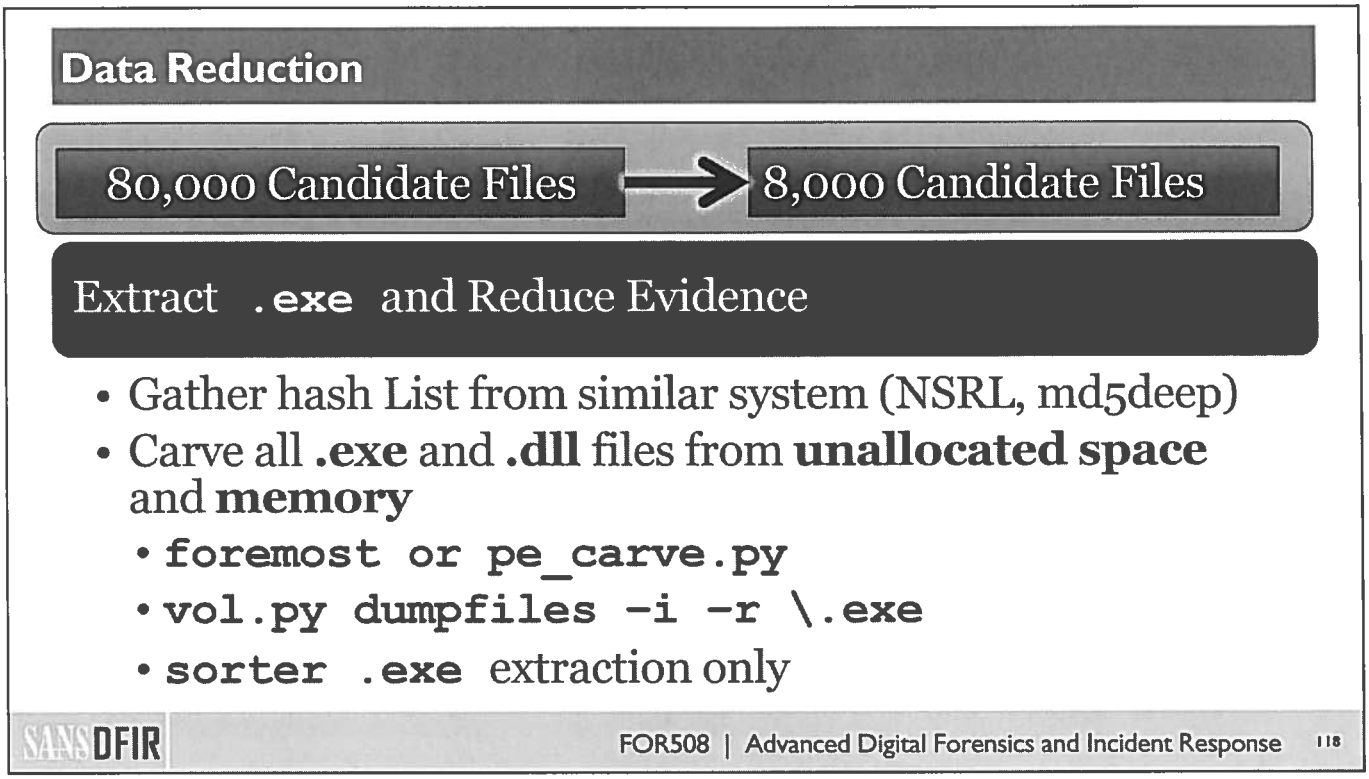


When hunting for a compromise, it is actually easiest if there is active malware on the system. In most cases, it gives you more places to look. Malware that is not active but is dormant is harder to detect as we lose the ability to detect the malware in memory. Malware that is dormant could launch through 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 if the system is compromised and malware doesn't currently exist. In this last situation, it is actually the hardest of the situations.

In the next section, we step through the three compromise situations. We will show you which of the hunting steps could be used as a "Detection Type" to help determine if a system is compromised under any of the three situations described above.



This page intentionally left blank.

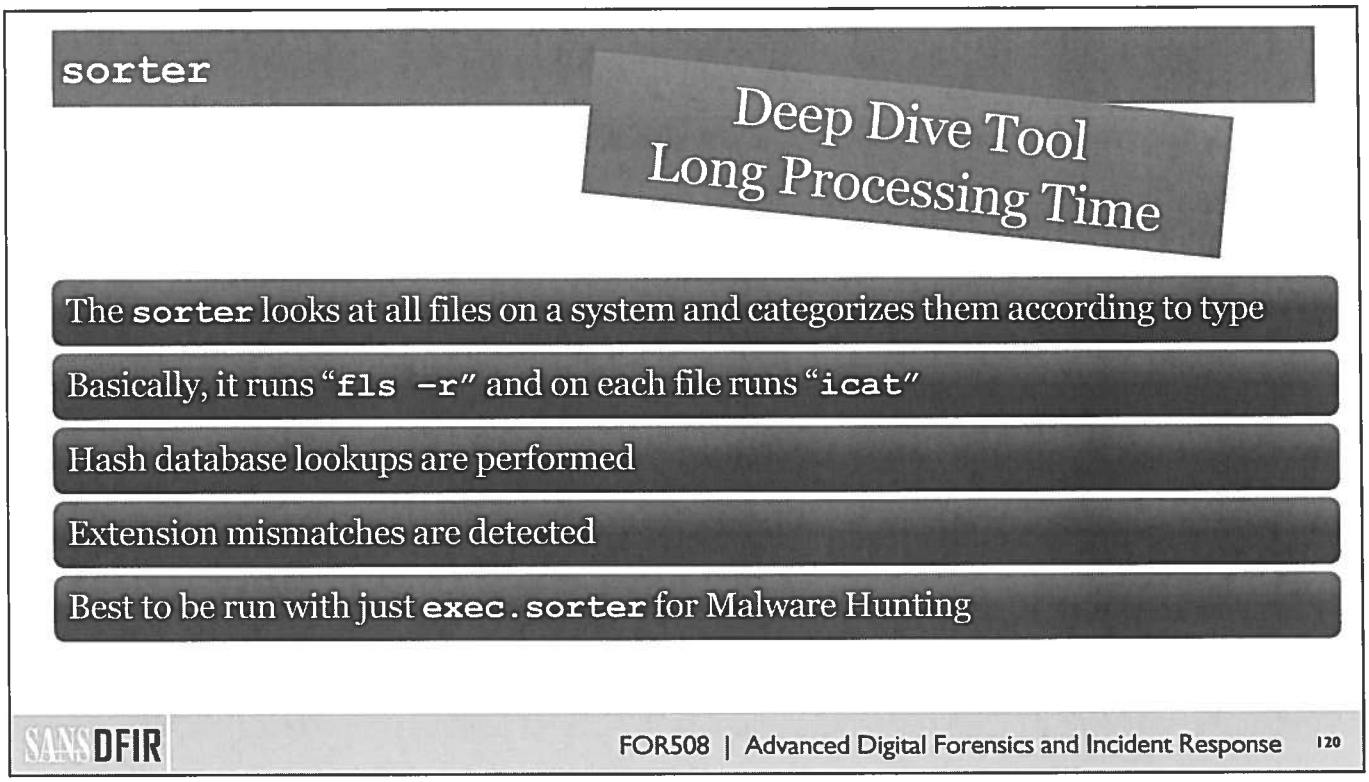


When you begin to look for malware on a machine, you first must have already carved out of the unallocated space as many intact .exe files as possible. You also should consider running sorter on the image to collect any files from the Metadata layer that are also still recoverable.

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

Data Reduction: Hash Comparisons

This page intentionally left blank.



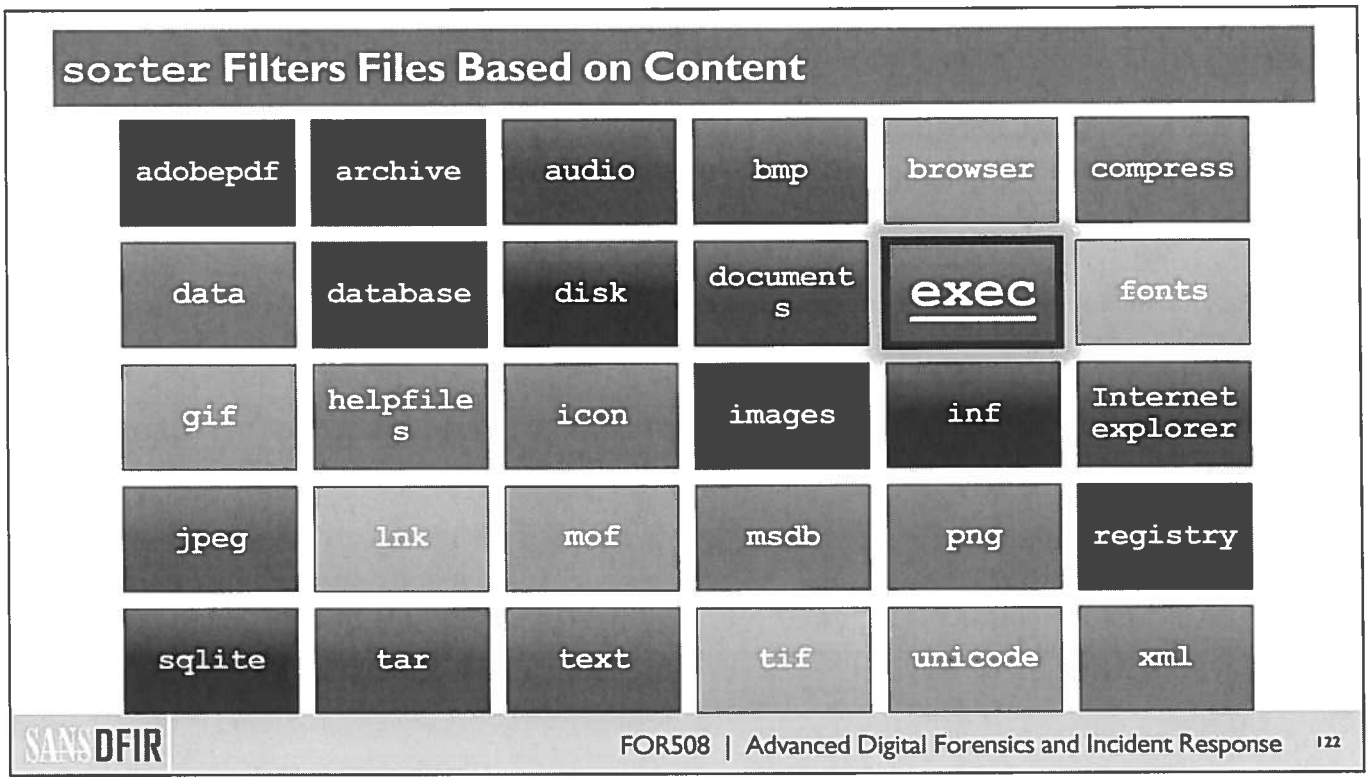
The sorter tool is a Perl script that combines many small tools from The Sleuth Kit. It runs the 'fls' tool to get a list of full file paths and then runs 'icat' on each. That data is sent to the 'file' command that identifies what file type it is. It then either saves the data to a directory or just makes a note of it. The tool can also look up into hash databases to see if the file is known. 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 useful.

Sorter takes the disk image as input and runs FLS against every file on the system. Not only will it run fls, but also 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 an .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.

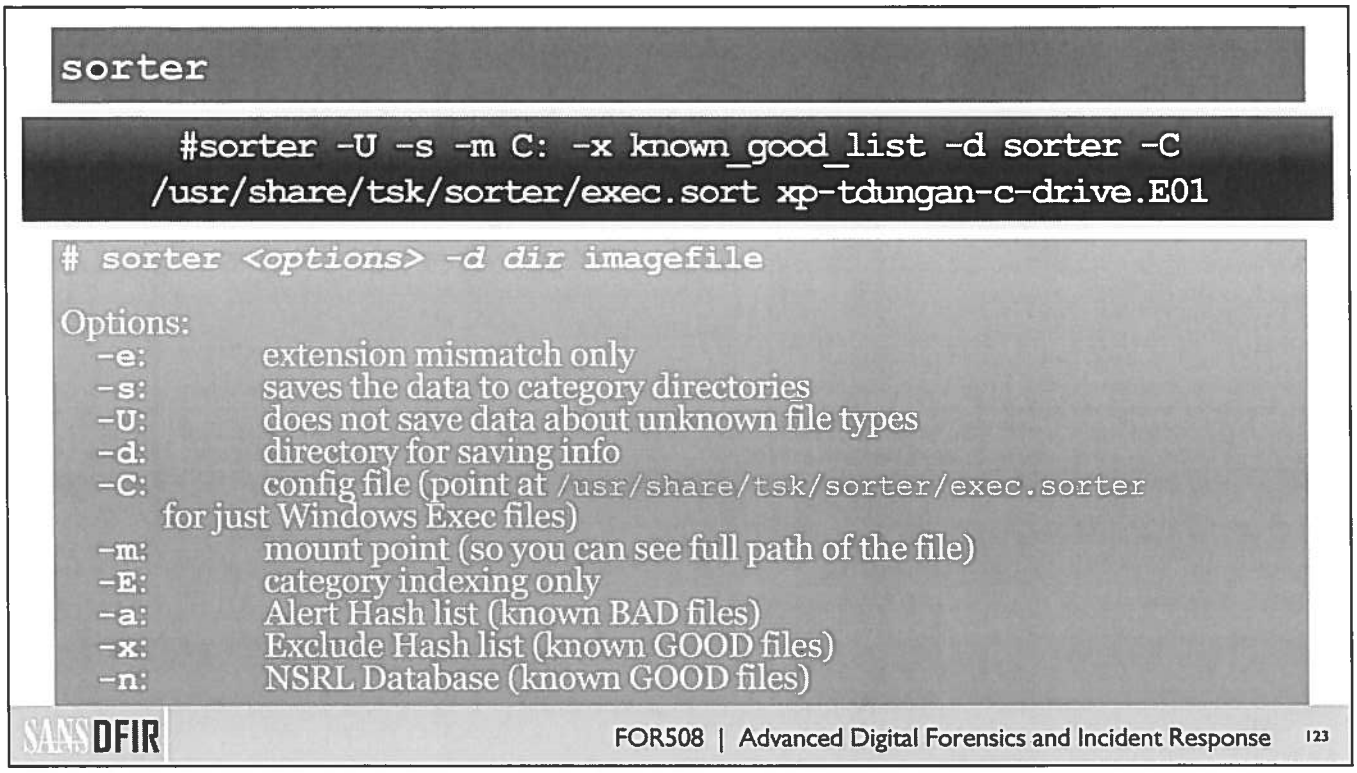
The sorter on SIFT has different configuration files than the default The Sleuth Kit distribution. We expanded the number of file type categories analyzed by sorter.

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 thumb drive.
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 change your perspective. The only downside to the tool is the length of time it needs to run. Plan for at least a couple hours for an executable file sort.



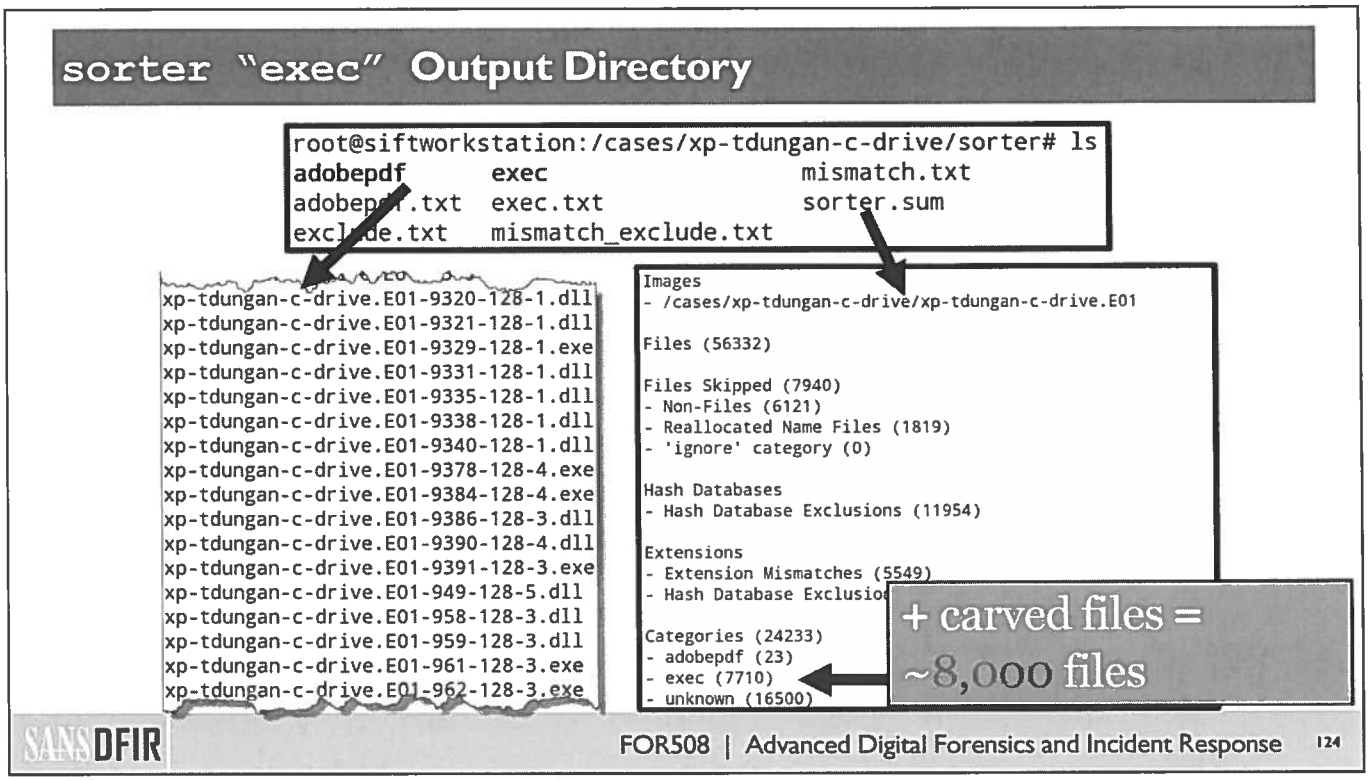

```
# hfind -i md5sum /cases/xp-tdungan-cdrive/precooked/hashes/WinXPSP3x86.txt
# sorter -U -s -m C: -x /cases/xp-tdungan-
cdrive/precooked/hashes/WinXPSP3x86.txt -d sorter -C
/usr/share/tsk/sorter/exec.sort /cases/xp-tdungan-c-drive/xp-tdungan-c-
drive.E01
```

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 want.

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.

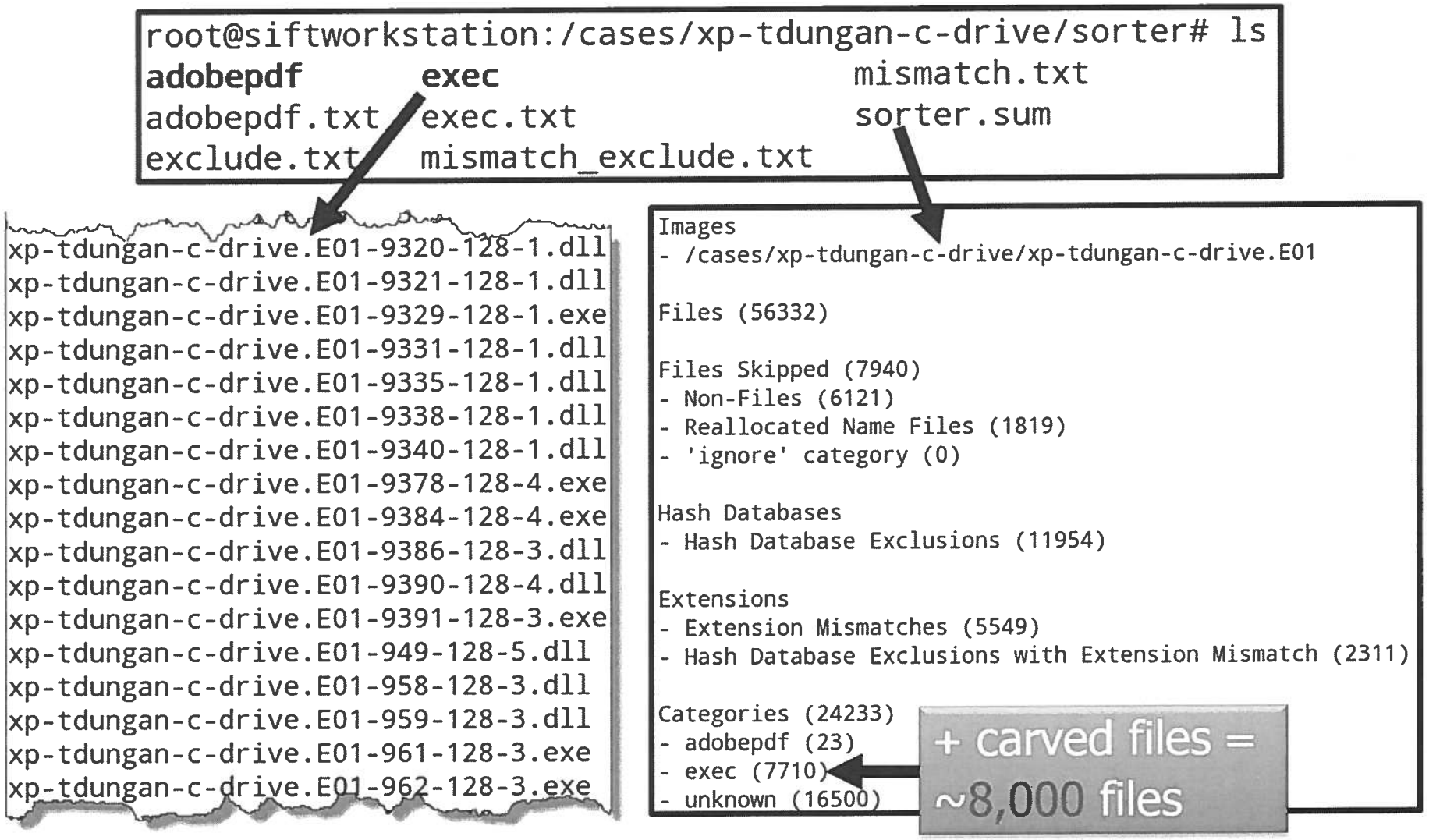
If the system is compromised, we can use the '-x' or the '-n' option of sorter, and all files that are found in the hash database will be excluded. Therefore, we will not see them in the sorter output, and we can focus on files that may have been created by the attack. We can also use the '–a' option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

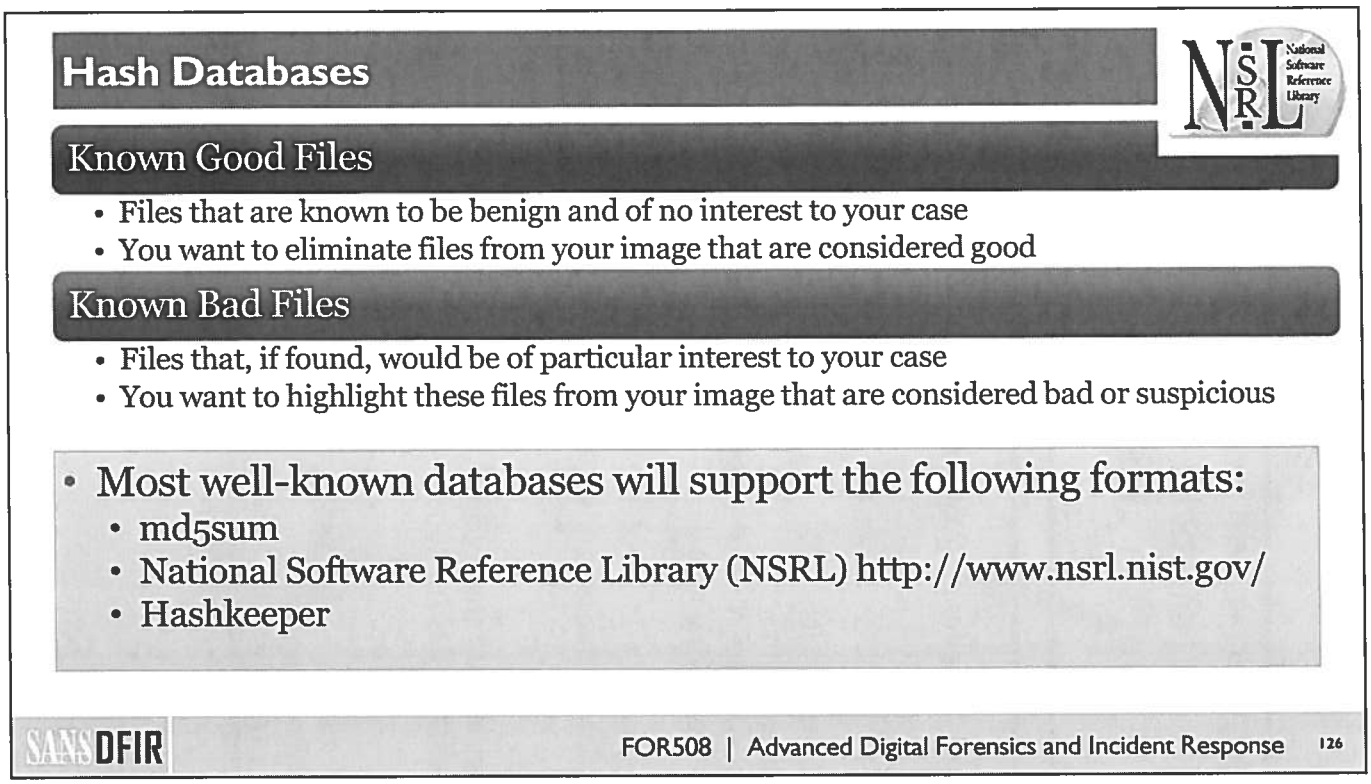
You will need to do your indexing and your collection for both known good and known evil hash lists. You will then use your sorter tool to be able to go through and alert on the hash list for known evil files, exclude hash list unknown good files, and the NSRL database.



This is the sorter output directory after the 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
adobepdf.txt
exclude.txt
exec
exec.txt
mismatch_exclude.txt
mismatch.txt
sorter.sum
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.

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

The National Software Reference Library is an arm of NIST (National Institute Science & Technology) and it has already done a little bit of this work for you. It has already created a known hash list of all the major operating systems and known applications. Now, the annoying thing 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 filesystem 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 by using an md5deep tool. The way it works is you run md5deep in recursive mode, add a starting directory, and redirect that down into a hash output file.

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

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

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.

Fuzzy Hashing

- Identifies similar files
  - Altered Documents
  - Partial Files
- Hashes and compares similarities between files based on hash values
  - Called Piecewise Hashing
- Typical hash algorithms take whole file
- Fuzzy hashing slices up files and examines smaller pieces

```
C:\>ssdeep ESSAY_DRAFT.doc
ssdeep,1.0--blocksize:hash:hash,filename
1536:py2WX5Q+30JmFKSG+xDg/h+xD+xD+xD+xD+xIWFxIWY+xD+xEwrpGvpGc+xD+xDn:Iz5Q0,"C:
ESSAY_DRAFT.doc"

C:\>ssdeep ESSAY_FINAL.doc
ssdeep,1.0--blocksize:hash:hash,filename
1536:xfgcSKWtiy538za73sKSG+xDg/h+xD+xD+xIWFxIWY+xD+xEwk+xD+xIWHAPuMx1:x5SKo5381
"C:\ESSAY_FINAL.doc"

C:\>ssdeep -b ESSAY_DRAFT.doc > hashes.txt

C:\>ssdeep -bm hashes.txt ESSAY_FINAL.doc
ESSAY_FINAL.doc matches ESSAY_DRAFT.doc (61)
```


One of the inaccuracies a responder would face is when he encounters a situation in which 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.

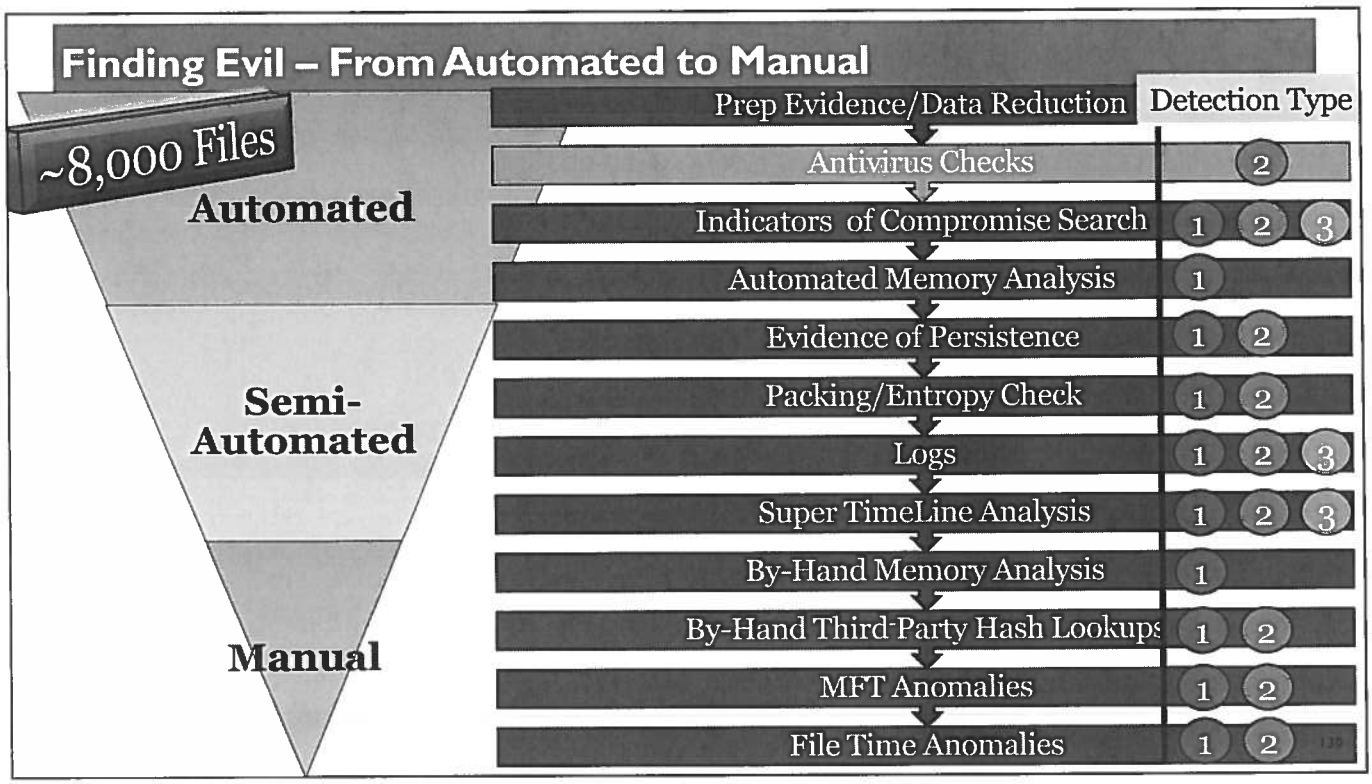
1. Collect a piecewise hash of a file or files and write it to a file.
   # ssdeep file.doc > file_hash.txt
2. Compare against similar file.
   #ssdeep –m file_hash.txt new_file.doc

Written by Jesse Kornblum

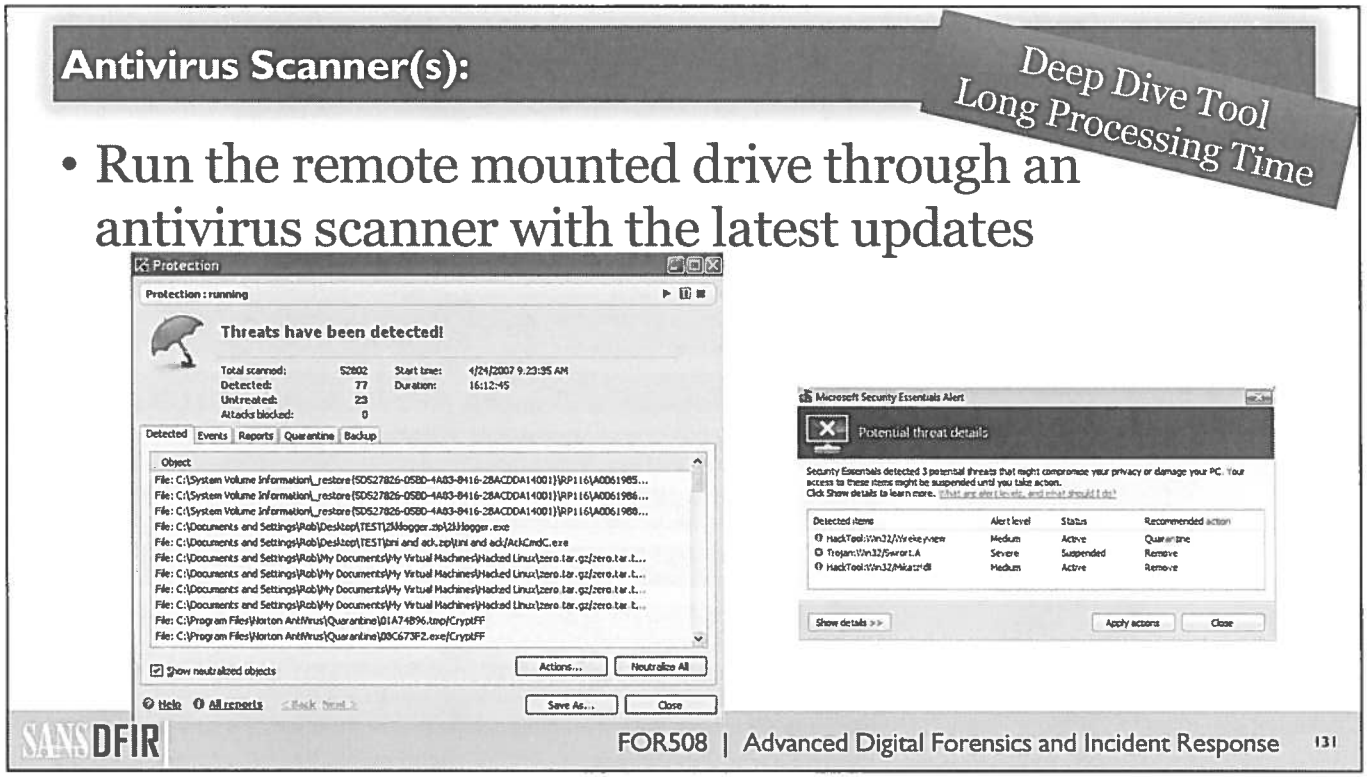
 # ssdeep -m *file_of_hashes* [OPTIONS] FILES

[Useful Options]
-m *file_of_hashes:* Load file of hashes for matching
-r: Enables recursive mode
-p: Pretty matching mode
-d: Enables 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

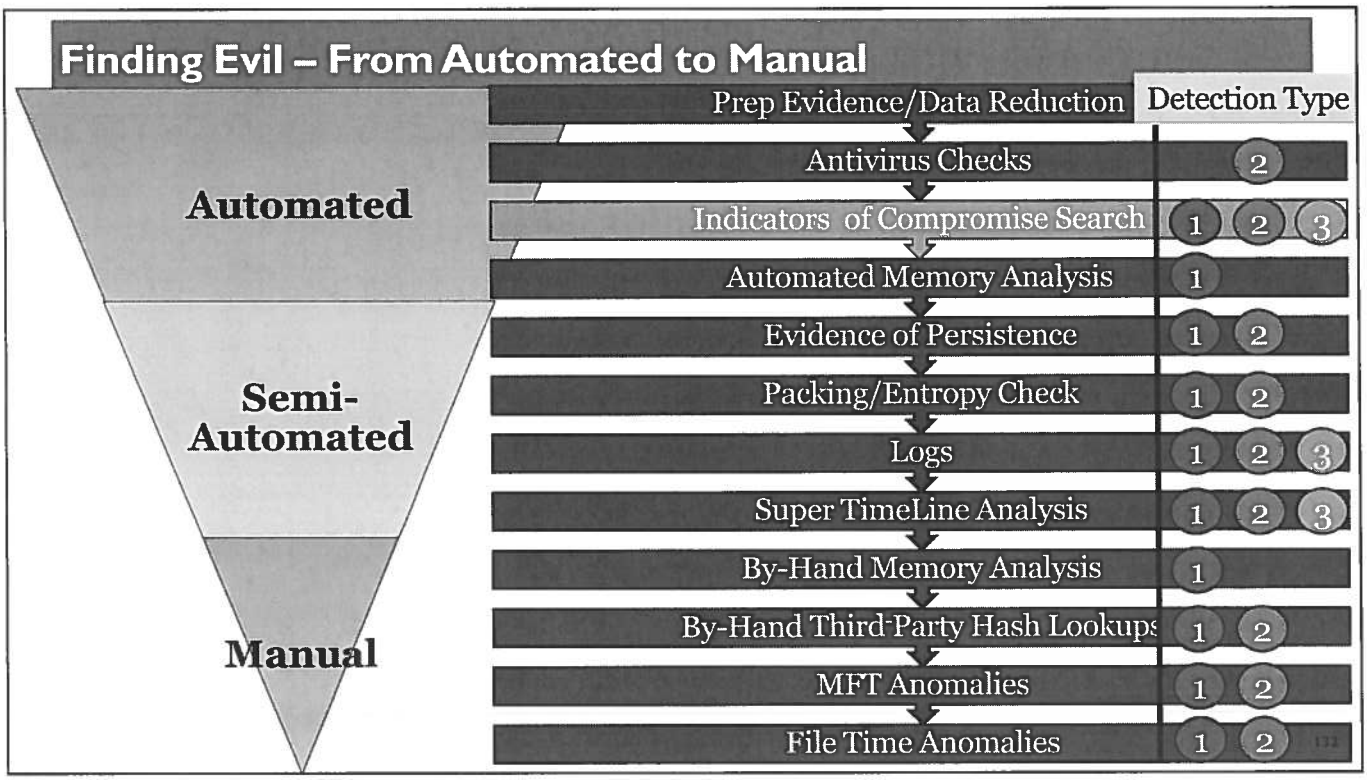


This page intentionally left blank.

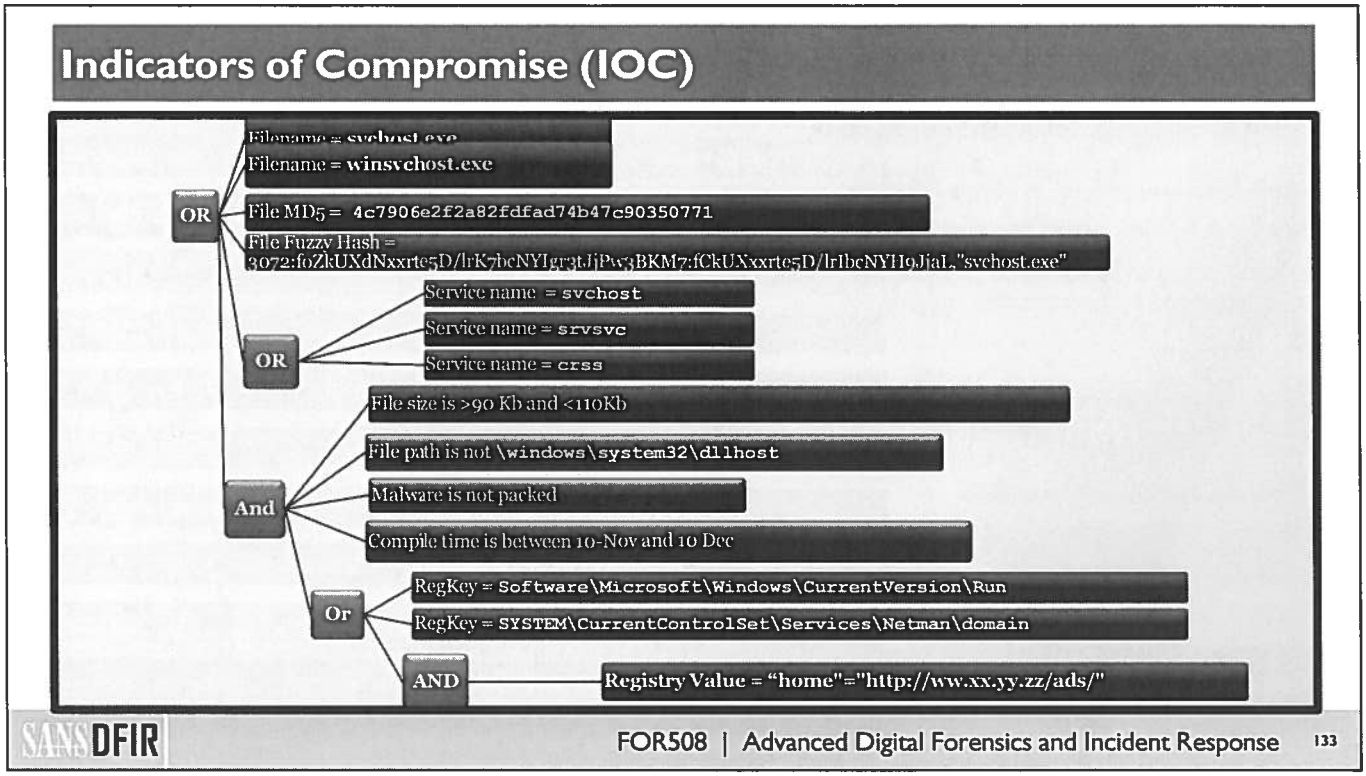


Antivirus scanners are also another tool you should quickly run on your system when you mount the drive for analysis. Antivirus scanners are also useful in incident response verification as they can quickly parse through your filesystem looking for malicious files. However, ensure that your antivirus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

Because 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, but I would also scan the sorter output directory to ensure you pick up any recovered deleted files during the sorter pass.



This page intentionally left blank.

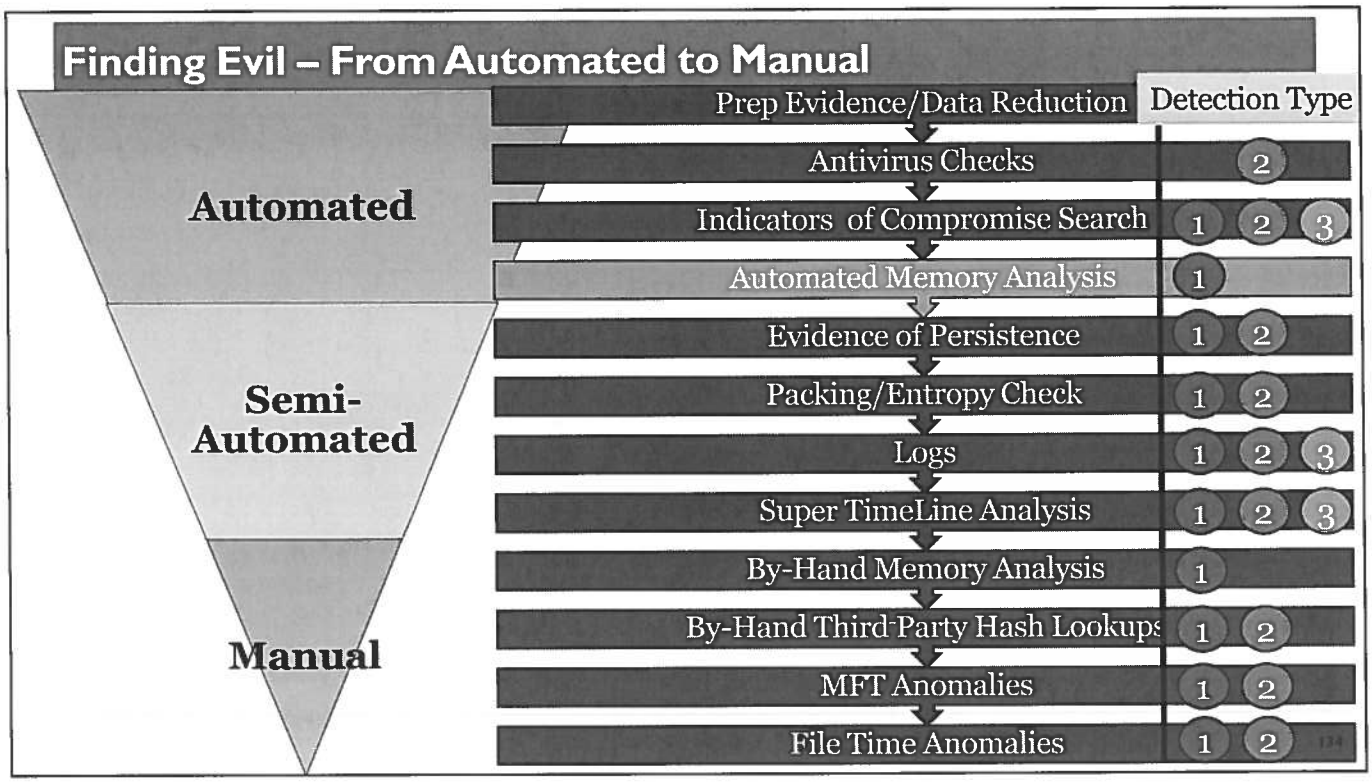


What is an indicator of compromise? It is a powerful technique to identify malware components on a compromised host. Generally, it is a combination of boolean expressions that can be used to identify general characteristics of malware. If these characteristics are found, then you have a hit.

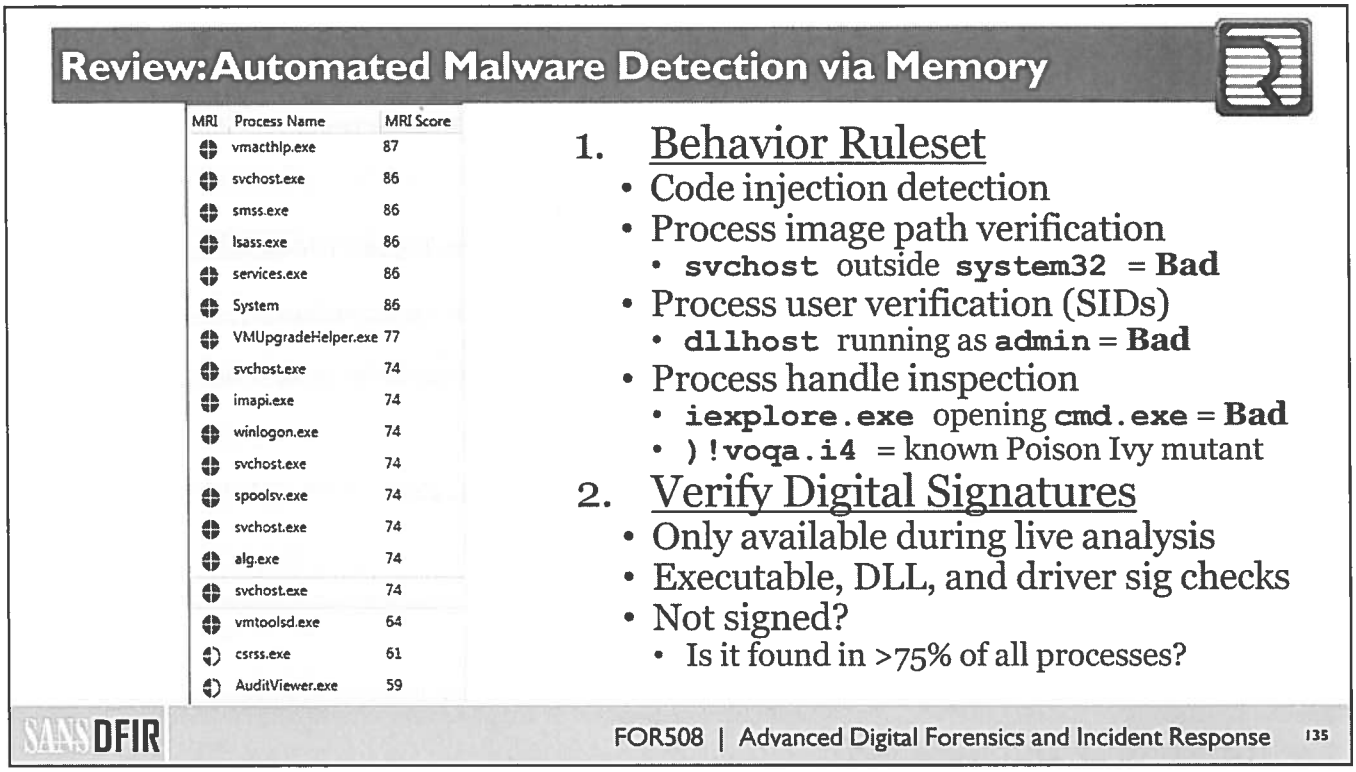
There are two types of indicators: Host-based (like you see above) and network-based (similar to snort signatures plus additional data).

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, after 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 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 nonsuspicious 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, and so on 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: behavior rule set and 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 be executed only from \windows\system32. Any time we see it running from somewhere else, we flag the process.

2. Process User - you can define a rule saying svchost.exe should be running as local service, network service, or system. When Redline sees svchost running as administrator, it gets flagged.

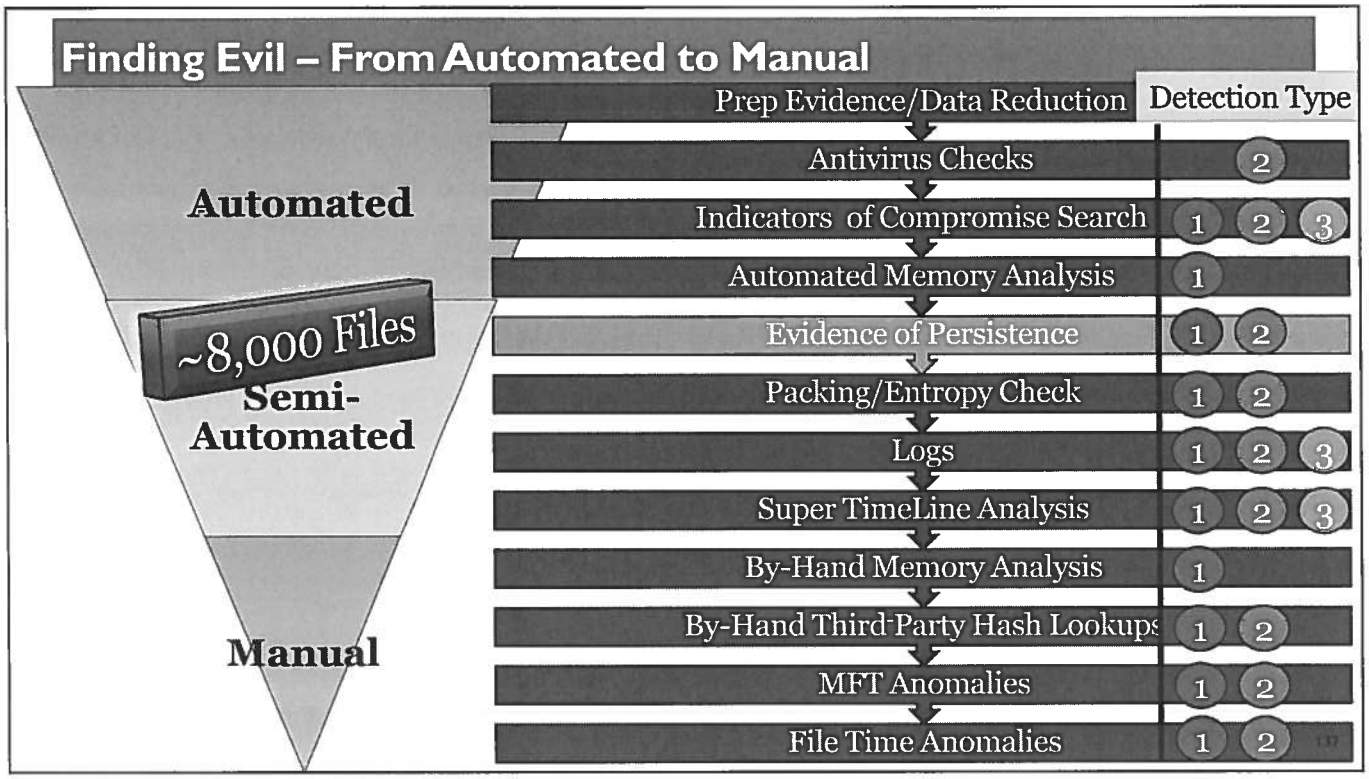
3. Process Handle Inspection allows you to define specific rules pertaining to malware or generic behavior. For example, a default rule is to flag svchost or iexplore anytime it has a process handle to cmd.exe. There is just no good reason for this to *ever* happen. You can also define rules based on 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 be enabled only 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.

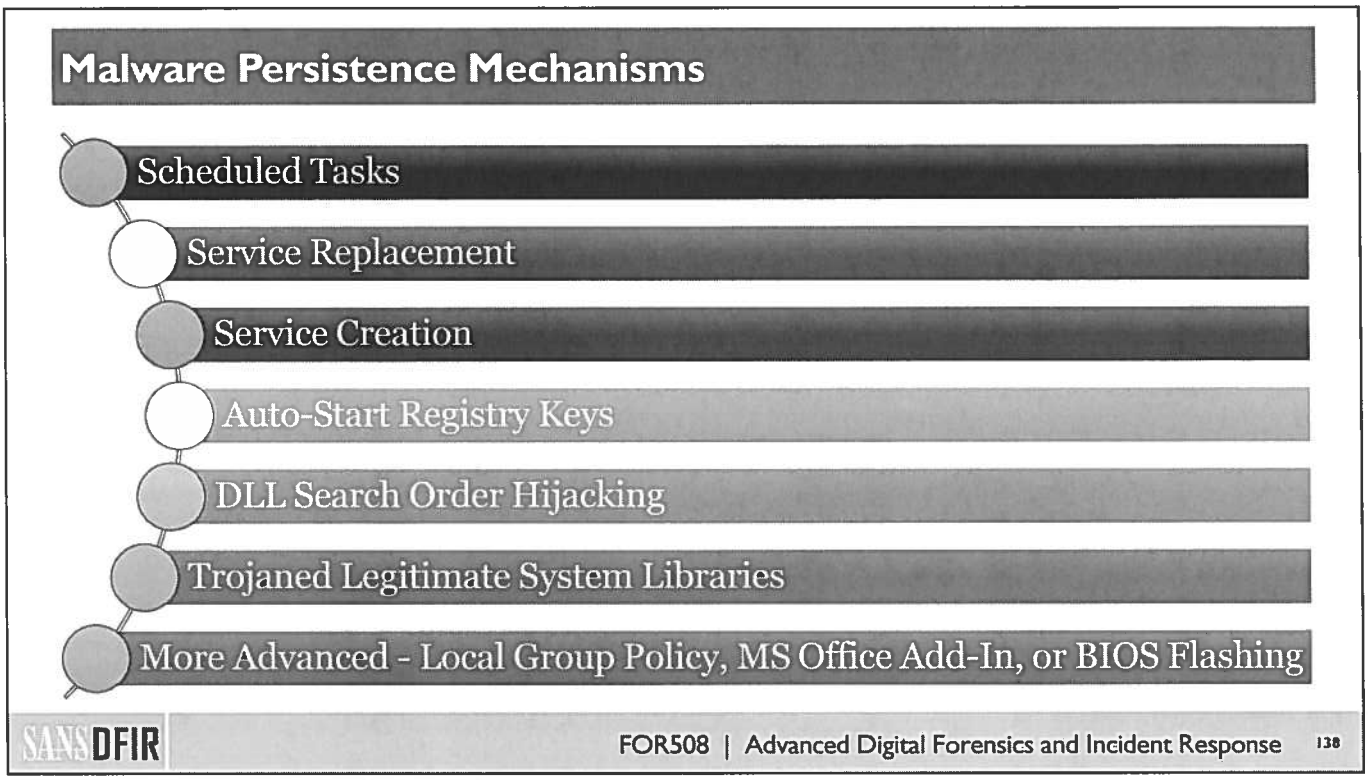
In addition to all the checks discussed in the Mandiant blog, Redline now also checks for the following:

- 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 need to know only 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, adversaries 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 created only 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

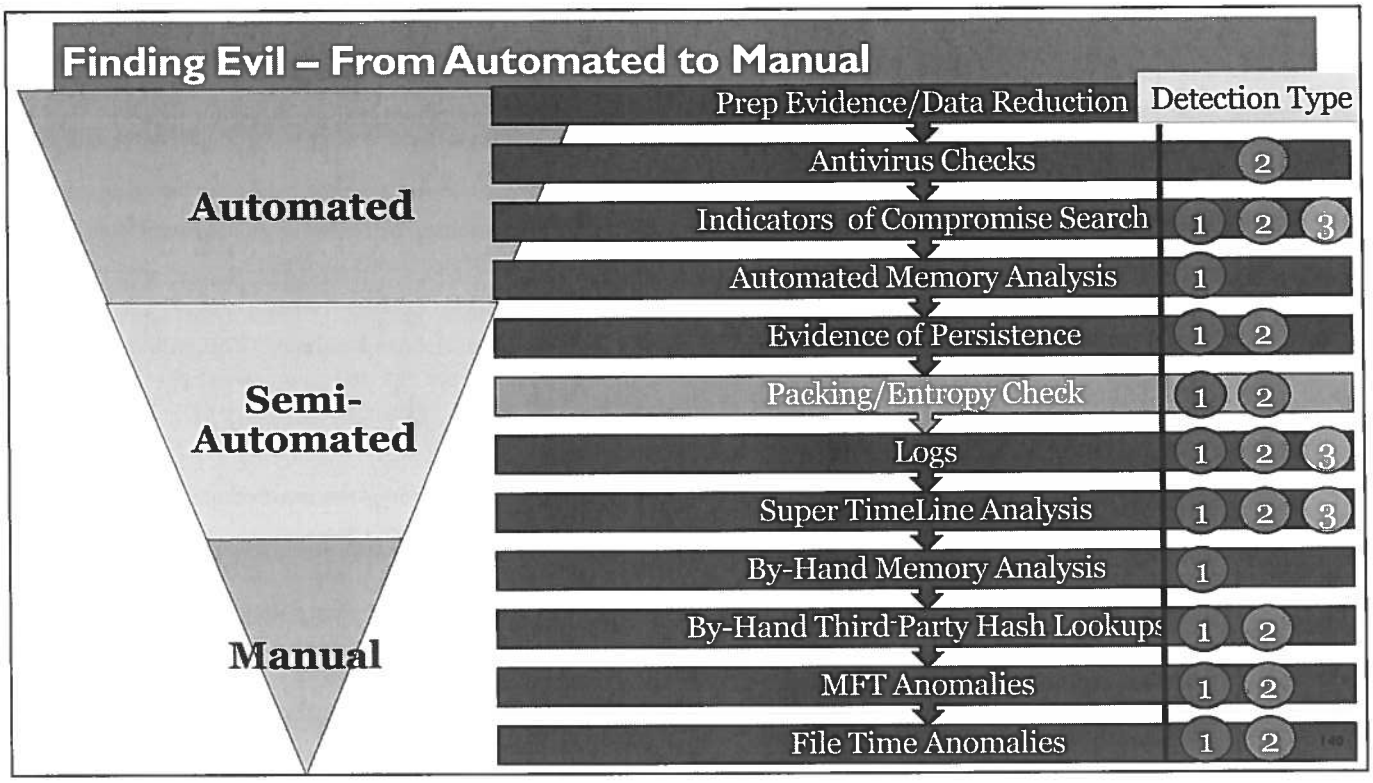
[3] https://blog.mandiant.com/archives/1207

[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/

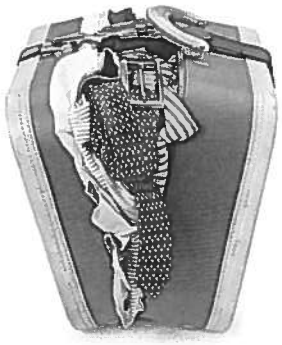


This page intentionally left blank.

Static Malware Identification: Files Trying to Hide Something

Tools to scan for malware
- Indications of packing
- Entropy
- Compiler and packing signatures
- Signed code

sigcheck
- Checks for signed code
- Upload to Virus Total



densityscout
- Checks for possible obfuscation and packing
- Files receive a score
- Score can be used to identify whether a set of files is worth further investigation

pescan
- Anomaly detection and scoring
- Binary section headers abnormal?
- Internal strings being obfuscated?
- Exports obfuscated?
- Resources contain possible embedded executable files?

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 these techniques are geared toward identifying how the Bad Guys got in, what they left behind, and whether 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 after 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 wants 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.

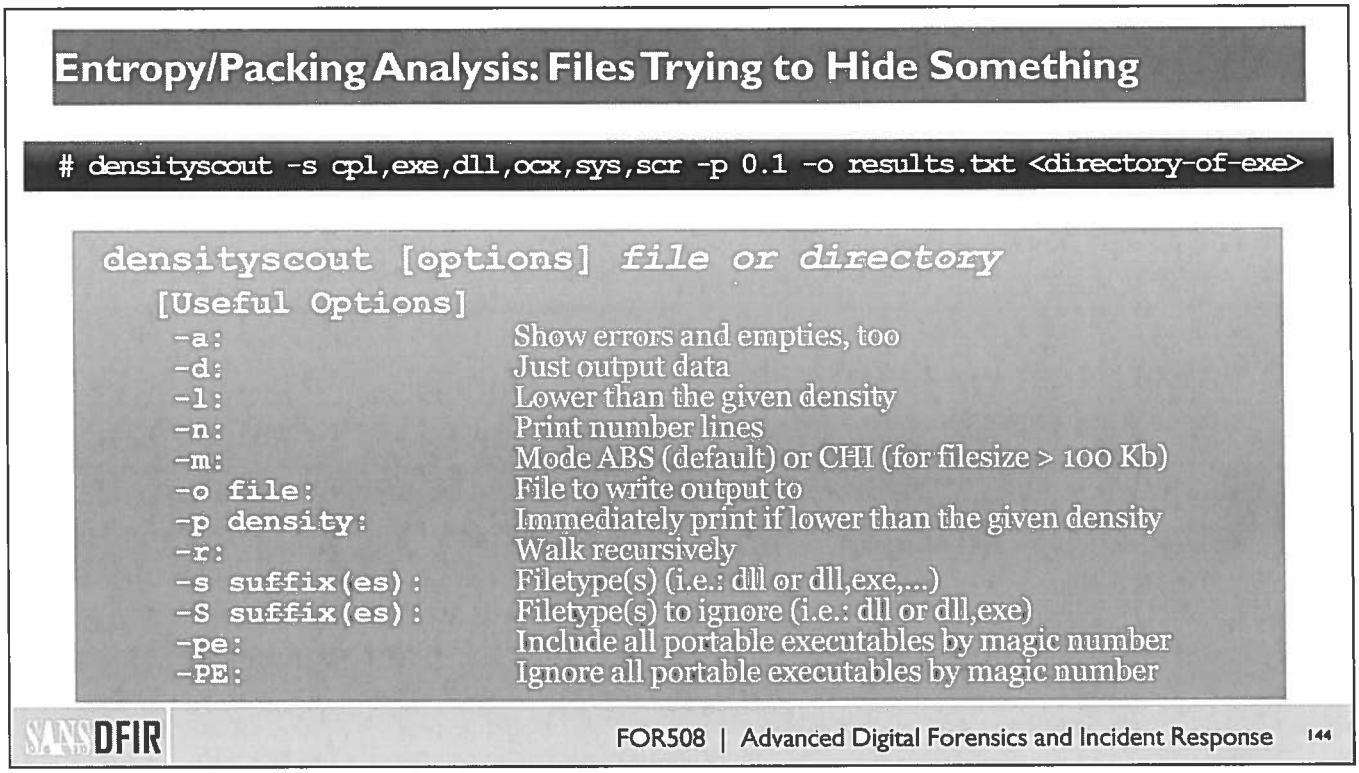
In addition to those hashing options, ***pescan*** has what we call “abnormality detection.” In the context of ***pescan***, this consists of looking at the PE file structure/packing and comparing it to what normally is available with a Windows operating system executable, dynamic link library, or driver 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 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 nonstandard compiler/linker. If there were 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?
- Does the binary make use of thread local storage?
- Is the import table present?
- Do 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?
- Were 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
```

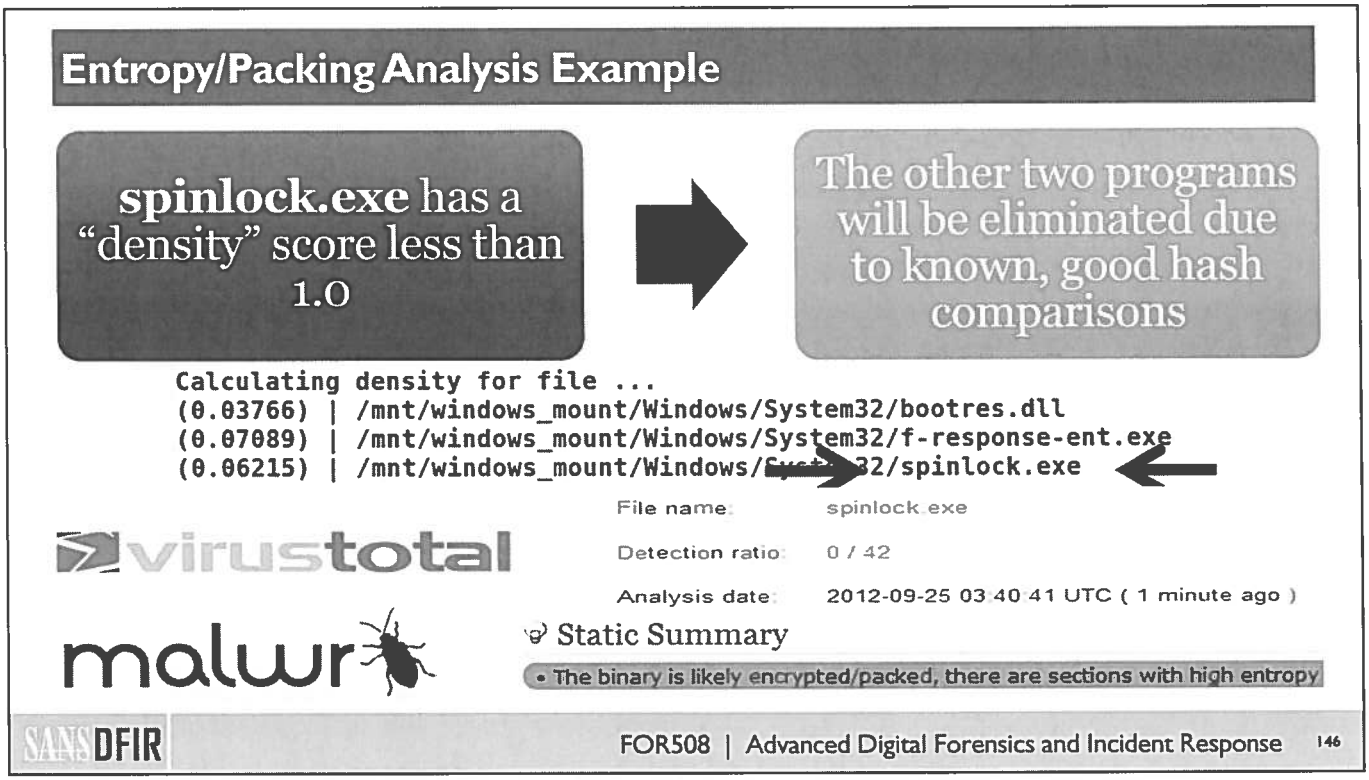
The option "-s cpl,exe,dll,ocx,sys,scr" tells DensityScout to include only files in further computations that have a typical portable executable extension. However, with the latest version (build 42) to achieve this kind of filtering, I recommend the all-new option "-pe." It tells DensityScout to select 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 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-brainer. 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 have to focus only on the top findings of it.

Last but not least, you specify the path to start from. In the current example it's only C:\Windows\System32 without any subdirectories, which is focused on. As mentioned this is one of the fastest approaches but a 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 1 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 1. However, with a quick md5 hash lookup, we can eliminate both bootres.dll and f-response-ent.exe from being suspicious.

Spinlock, however, shows up not as a known file at all but based on the density makes it more suspicious. If spinlock were 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
```

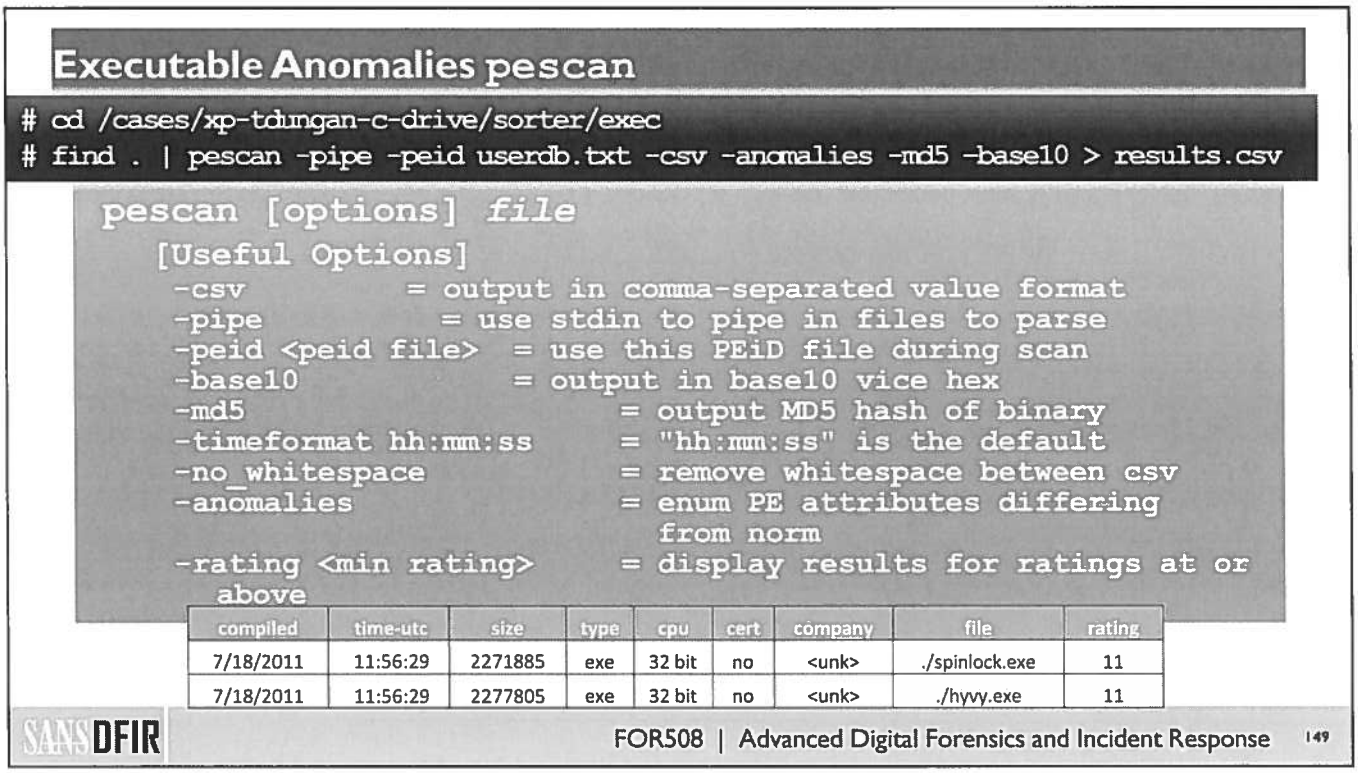
File name: spinlock.exe

Detection ratio: 0 / 42

Analysis date: 2012-09-25 03:40:41 UTC (1 minute ago)

Static Summary

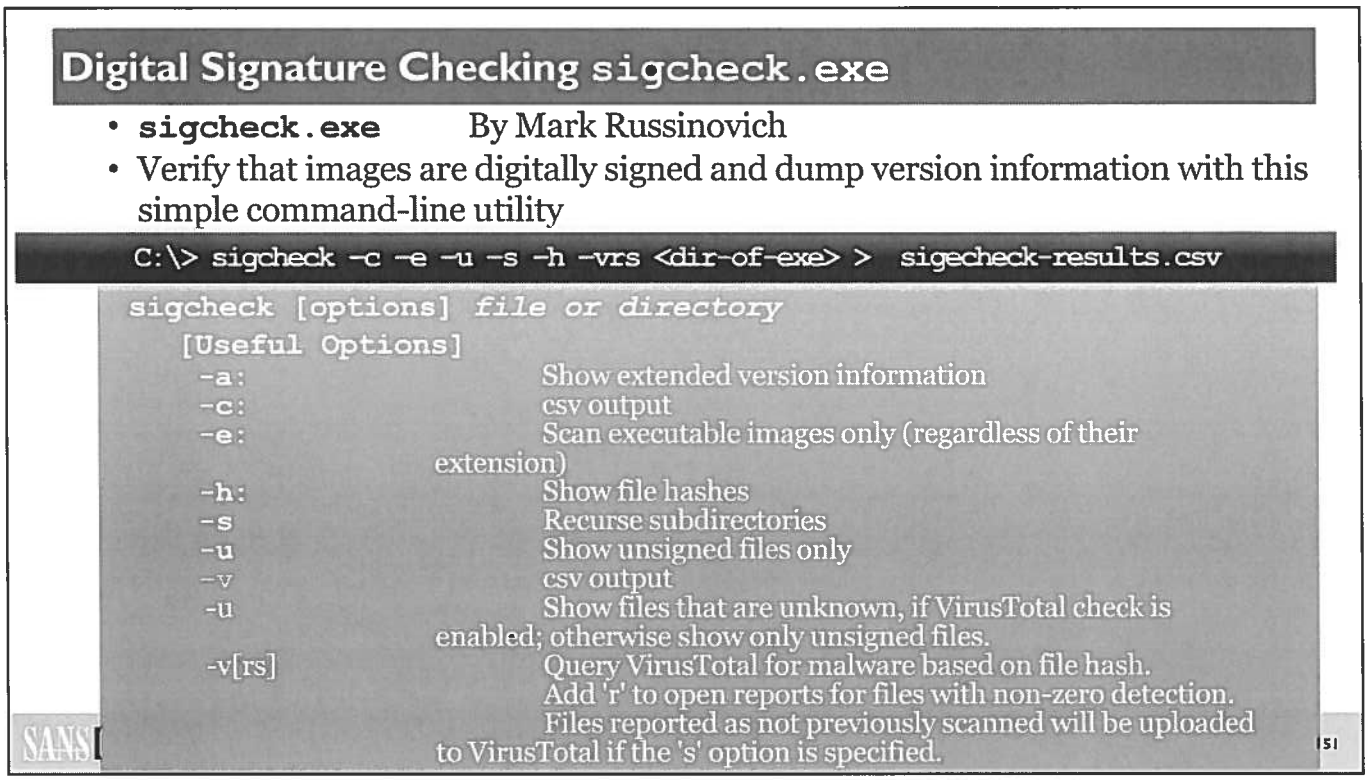
- The binary is likely encrypted/packed, there are sections with high entropy




```
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 9	11:56:22	22778 05	exe	32 bit	Win2K	no	yes	<unk>	./xp-tdungan-c-drive.E01-**5237**-128-4.exe	11
7/18/2011 9	11:56:22	22718 85	exe	32 bit	Win2K	no	yes	<unk>	./xp-tdungan-c-drive.E01-**7793**-128-3.exe	11
10/24/2011 4	21:04:12	20008 96	dll	32 bit	NT4	no	yes	Apple Inc.	./xp-tdungan-c-drive.E01-27092-128-4.qtx	10
2/22/2012 0	21:32:32	22067 2	dll	32 bit	NT4	yes	yes	<unk>	./xp-tdungan-c-drive.E01-13061-128-4.dll	9
2/7/2012 3	22:58:12	17474 56	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.

Although, 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 your code, you can make it more difficult to find; however, when found, you expose every system currently infected with it after the cert has been revoked.

What is 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.

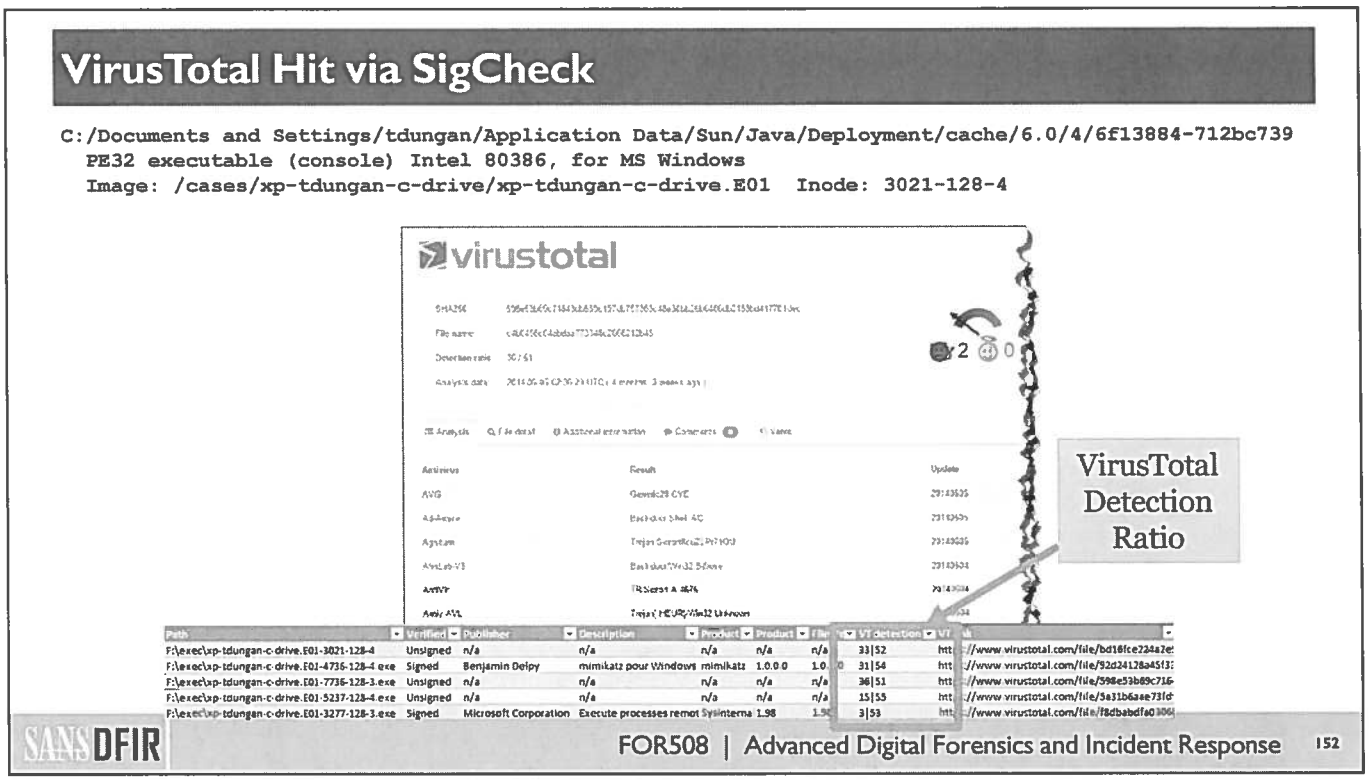
As a result, it is likely that much of the malware you will encounter will not be signed. However, the examples of signed malware will continue to increase over time as more advanced malware will be discovered. Using a simple tool like sigcheck can help you easily find the unsigned executables in a directory. If you combine this with densityscout and elimination of known good hashes using sorter, it is likely you will have 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 antivirus 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 I recommend that you run this against your filtered, sorter directory after you have removed known_good hashes.

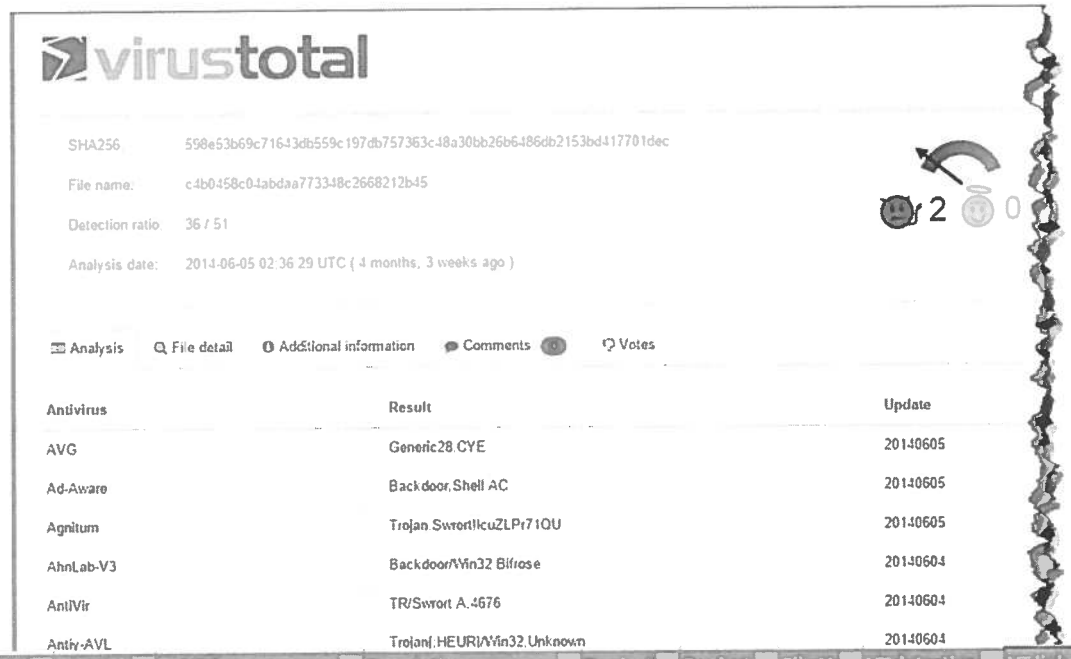
C:/Documents and Settings/tdungan/Application Data/Sun/Java/Deployment/cache/6.0/4/6f13884-712bc739
PE32 executable (console) Intel 80386, for MS Windows
Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01 Inode: 3021-128-4
MD5: 7a0fcd15ee1c2d9d196ab6515adf2f87
Saved to: exec/xp-tdungan-c-drive.E01-3021-128-4

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

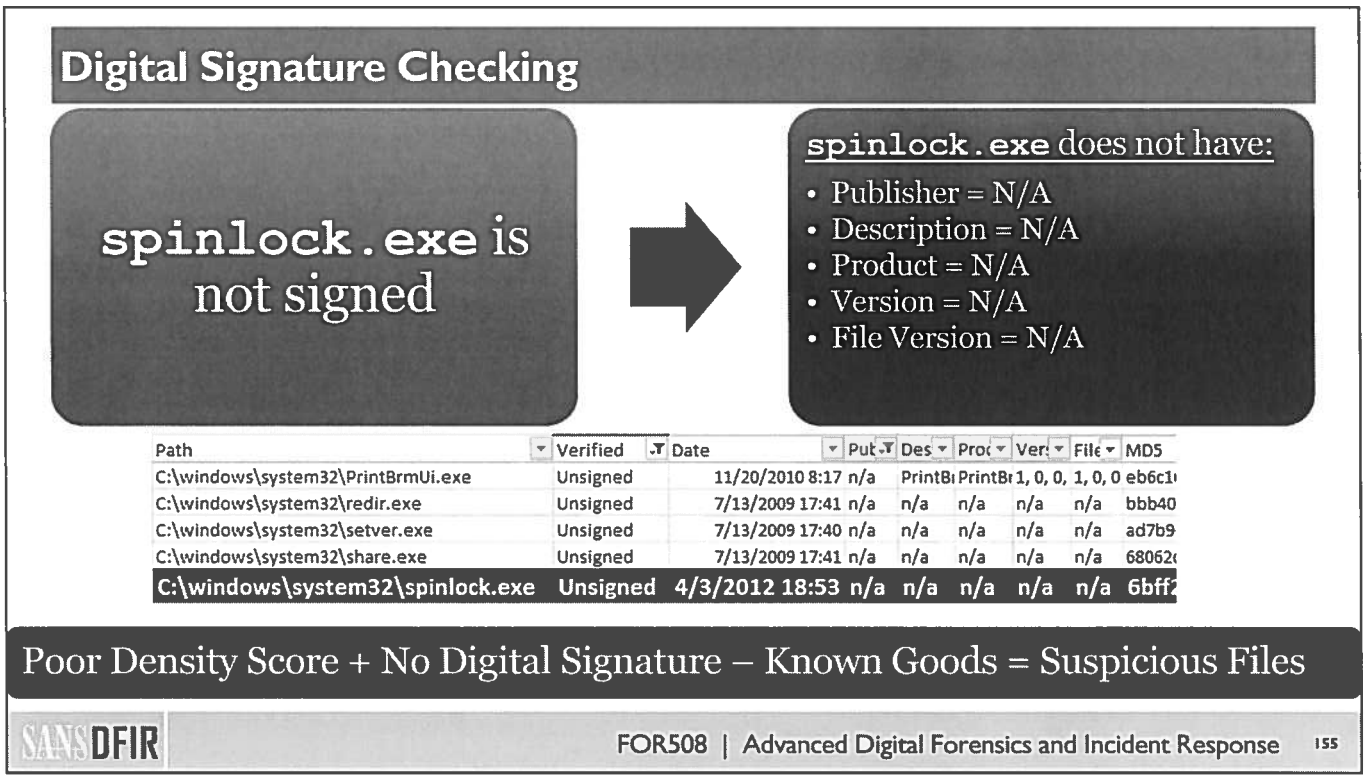
C:/WINDOWS/system32/hyvy.exe
PE32 executable (GUI) Intel 80386, for MS Windows
Image: /cases/xp-tdungan-c-drive/xp-tdungan-c-drive.E01 Inode: 5237-128-4
MD5: 0159fbbdaff9291425ab976dbbd40aa0
Saved to: exec/xp-tdungan-c-drive.E01-5237-128-4.exe

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

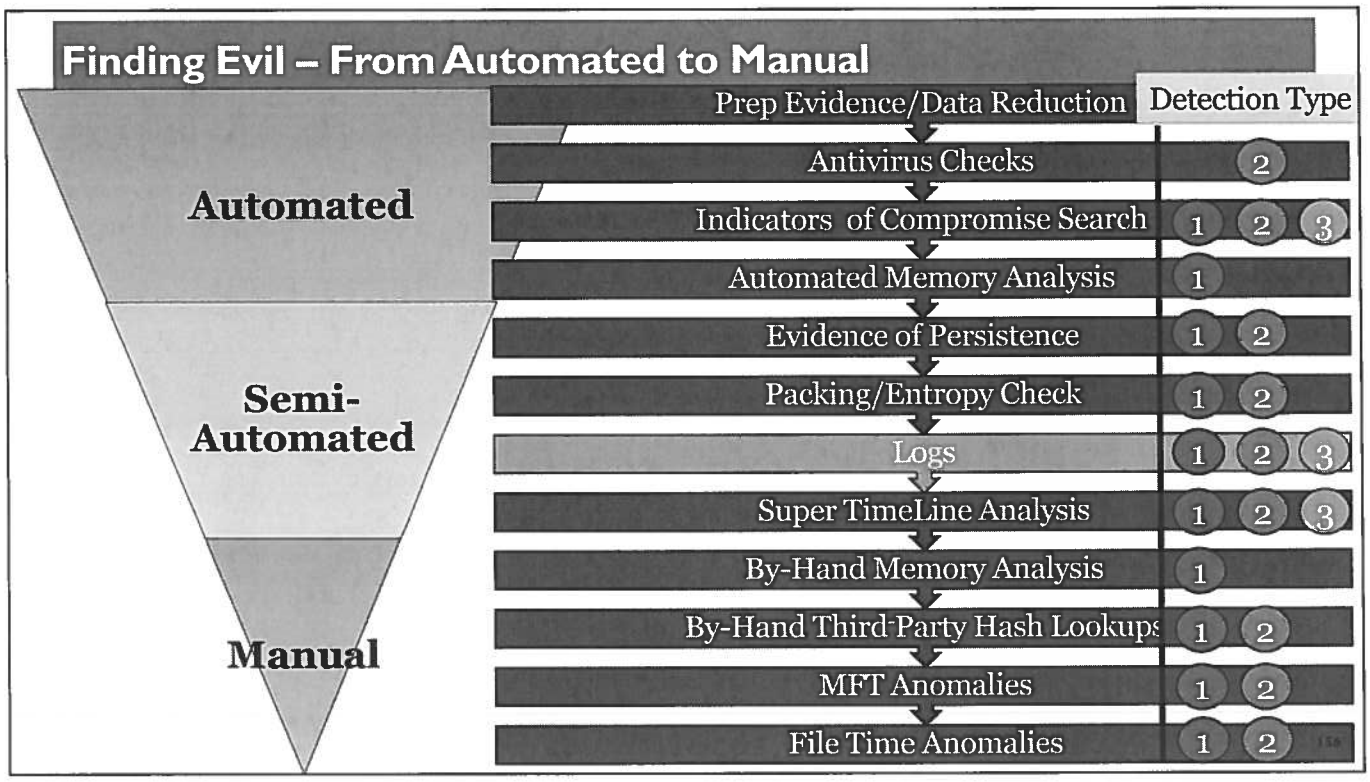


Path	Verified	Publisher	Description	Product	Product	File Ve	VT detection	VT link
F:\exec\xp-tdungan-c-drive.E01-3021-128-4	Unsigned	n/a	n/a	n/a	n/a	n/a	33|52	https://www.virustotal.com/file/bd16fce224a2e
F:\exec\xp-tdungan-c-drive.E01-4736-128-4.exe	Signed	Benjamin Delpy	mimikatz pour Windows	mimikatz	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

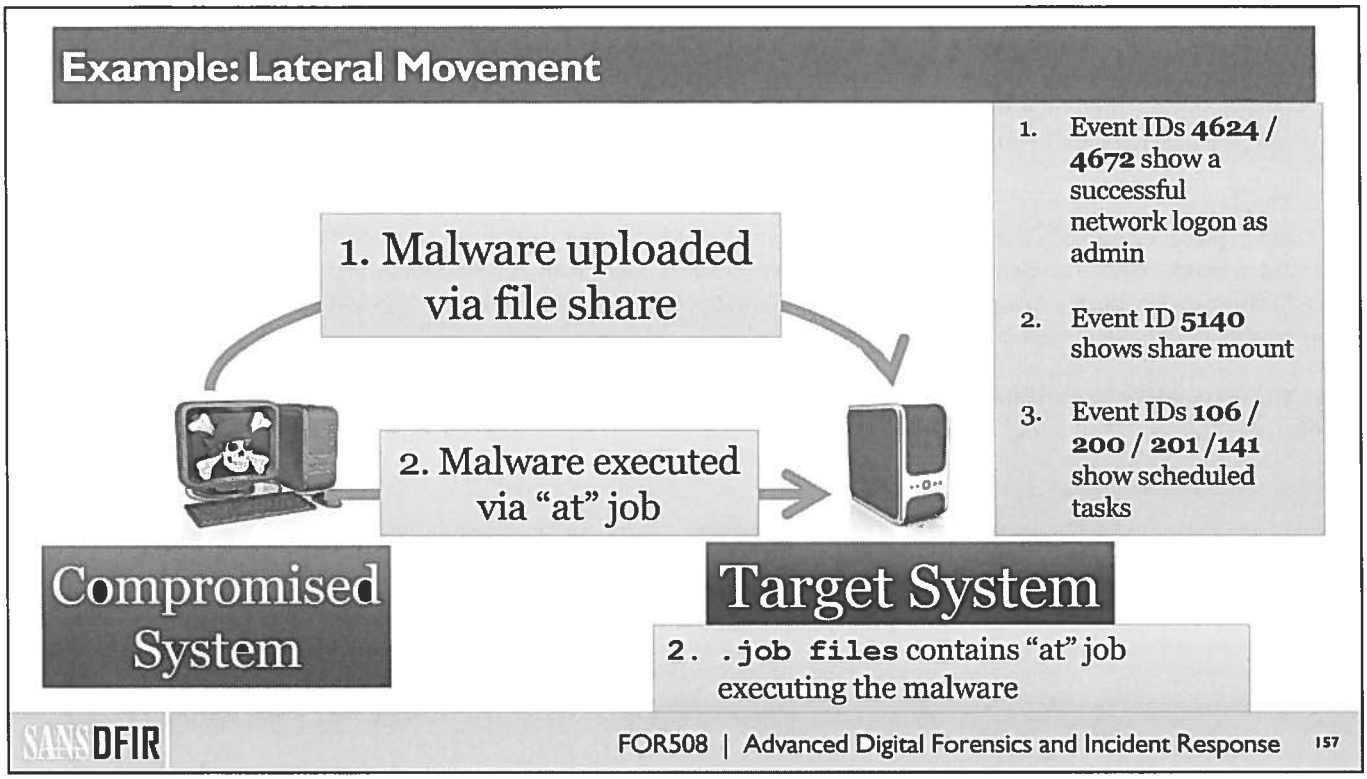


For spinlock, it is already suspicious it had a low entropy/density score. But the file is also not signed. If you subtracted out all the known-good files from the same directory, spinlock would look fairly alone.

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. Although not unique as there is valid software that also doesn't follow this practice, in my experience approximately 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.



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 to move through a domain as quietly as possible to avoid logging on the domain controller.

After 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. After the malware has been successfully copied to the remote machine, an "at" or "schtask" 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 a responder 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 attackers have 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 be present only 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 an 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. Commonly a simple action by a user can cause multiple event log entries to be recorded throughout the network. When reviewing Event Logs in a networked environment, it is 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

Example: Lateral Movement Timeline

Security.evtx	
12:00:00	4624 – Network logon
12:00:00	4672 – Admin rights
12:00:15	5140 – Network share
Microsoft-Windows-TaskScheduler%4Operational.evtx	
12:02:23	106 – Task scheduled
12:03:00	200 – Task executed
12:03:12	201 – Task completed
12:05:41	141 – Task removed
Security.evtx	
12:07:01	4634 – Logoff

One of the best ways to learn how lateral movement can be tracked via event logs is through examples. In this example, we will follow a sequence of events and determine the answers to key questions along the way.

Key questions that can be answered:

Login events

- *4624 – Who, where from, time 4672 – They're an admin*
- *5140 – And they mounted a share from …*

Scheduled tasks

- *106 – Job name, who, time*
- *200 – Start time and program name 201 – Finish time*
- *141 – They cleaned up*

4624 – Network Logon – Secdurity.evtx

- Timestamp = 2015-02-14 12:00:00
- Event ID = 4624
- TargetUserSid = S-1-5-21-2723264887-207281631-482592677-2984
- TargetUserName = imowned

- TargetLogonId = 0x000000021457dbab
- LogonType = 3
- LogonProcessName = Kerberos
- AuthenticationPackageName = Kerberos
- LogonGuid = {726F6B9E-C1BE-4EC1-BB95-3B0B6238BE56}
- IpAddress = 10.1.1.10

4672 – Admin Rights – Security.evtx

- Timestamp = 2015-02-14 12:00:00
- Event ID = 4672
- SubjectUserName = imowned
- SubjectDomainName = MYDOM
- SubjectLogonId = 0x000000021457dbab

5140 – Network Share – Security.evtx

- Timestamp = 2015-02-14 12:00:15
- Event ID = 5140
- SubjectUserSid = S-1-5-21-2723264887-207281631-482592677-2984
- SubjectUserName = imowned
- SubjectDomainName = MYDOM
- ObjectType = File
- IpAddress = 10.1.1.10
- ShareName = //*/C$
- ShareLocalPath = /??/C:/

106 – Task Scheduled - Microsoft-Windows-TaskScheduler%4Operational.evtx

- Timestamp = 2015-02-14 12:02:23
- Event ID = 106 TaskName = /At1
- UserContext = MYDOM/imowned

200 – Task Executed - Microsoft-Windows-TaskScheduler%4Operational.evtx

- Timestamp = 2015-02-14 12:03:00
- Event ID = 200
- TaskName = /At1
- ActionName = malicious.bat
- TaskInstanceId = {B042B2E4-D186-4970-AE6C-B5DE328BCF3A}

201 – Task Completed - Microsoft-Windows-TaskScheduler%4Operational.evtx

- Timestamp = 2015-02-14 12:03:12
- Event ID = 201

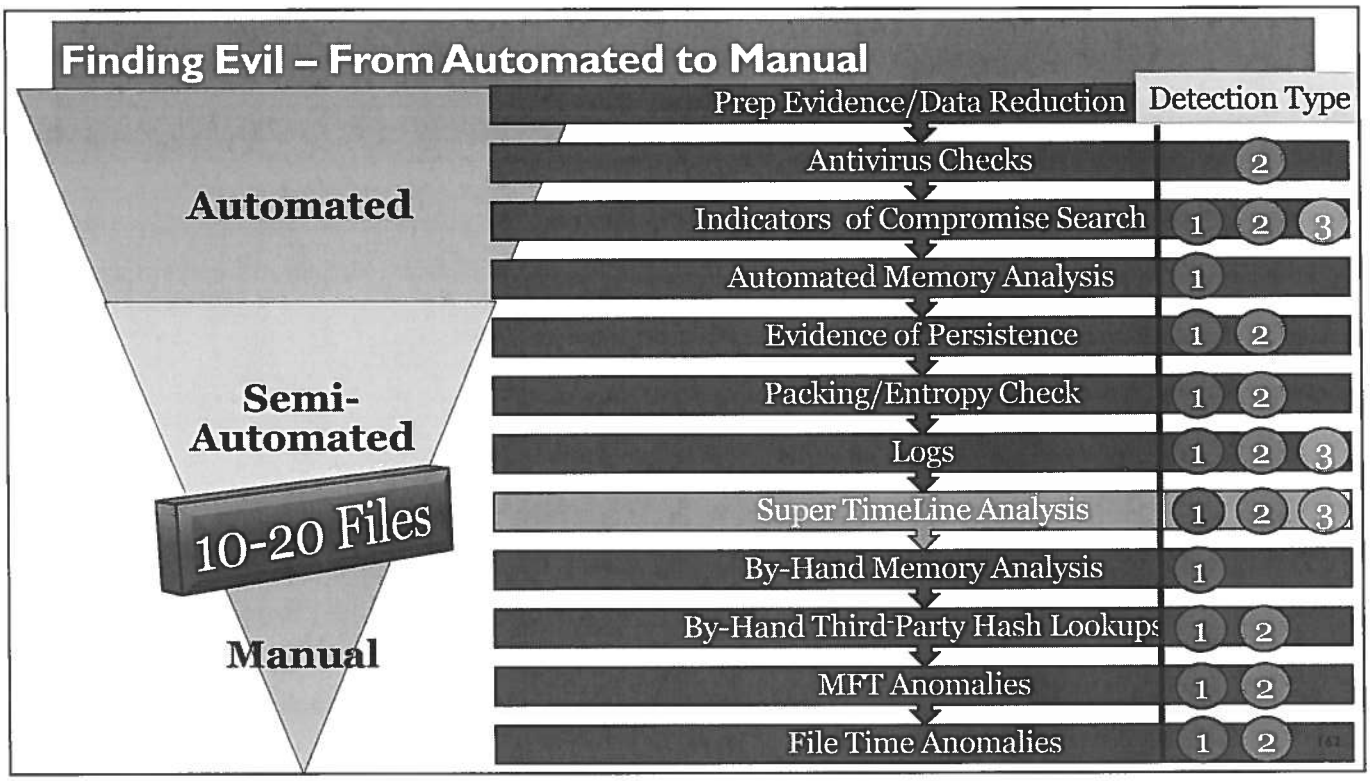
- TaskName = /At1
- TaskInstanceId = {B042B2E4-D186-4970-AE6C-B5DE328BCF3A}
- ActionName = C:/Windows/SYSTEM32/cmd.exe
- ResultCode = 0

141 – Task Removed - Microsoft-Windows-TaskScheduler%4Operational.evtx

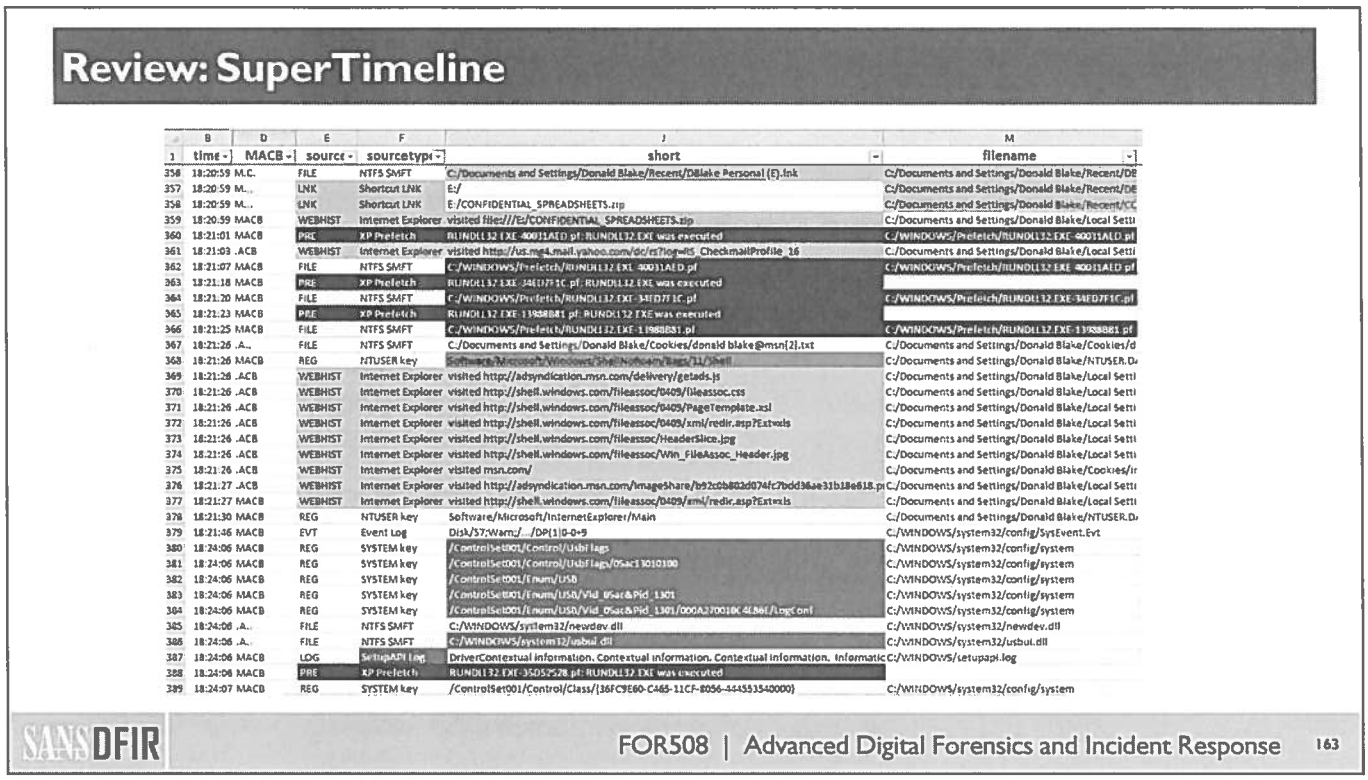
- Timestamp = 2015-02-14 12:05:41
- Event ID = 141
- TaskName = /At1
- UserName = NT AUTHORITY/System

4634 – Logoff

- Timestamp = 2015-02-14 12:07:01
- Event ID = 4634
- TargetUserSid = S-1-5-21-2723264887-207281631-482592677-2984
- TargetUserName = imowned
- TargetDomainName = MYDOM
- TargetLogonId = 0x000000021457dbab
- LogonType = 3

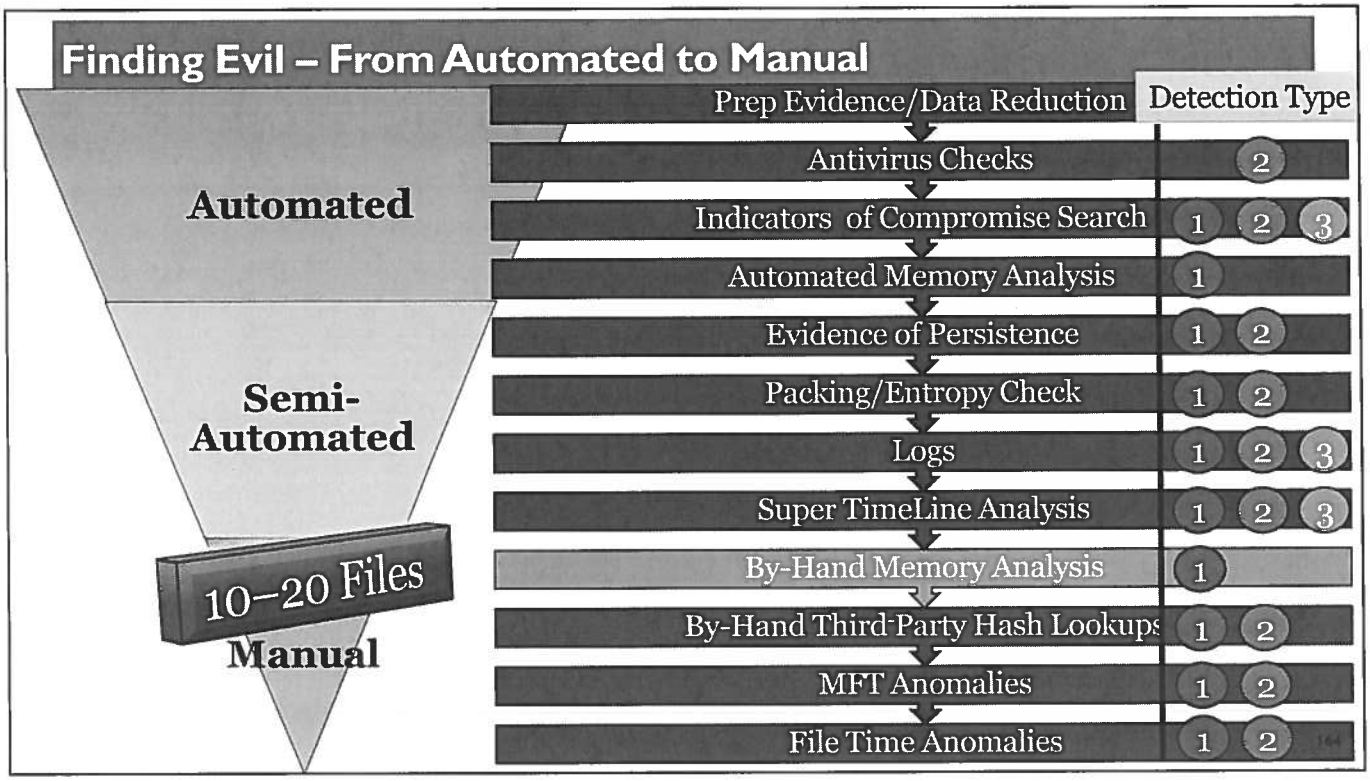


This page intentionally left blank.

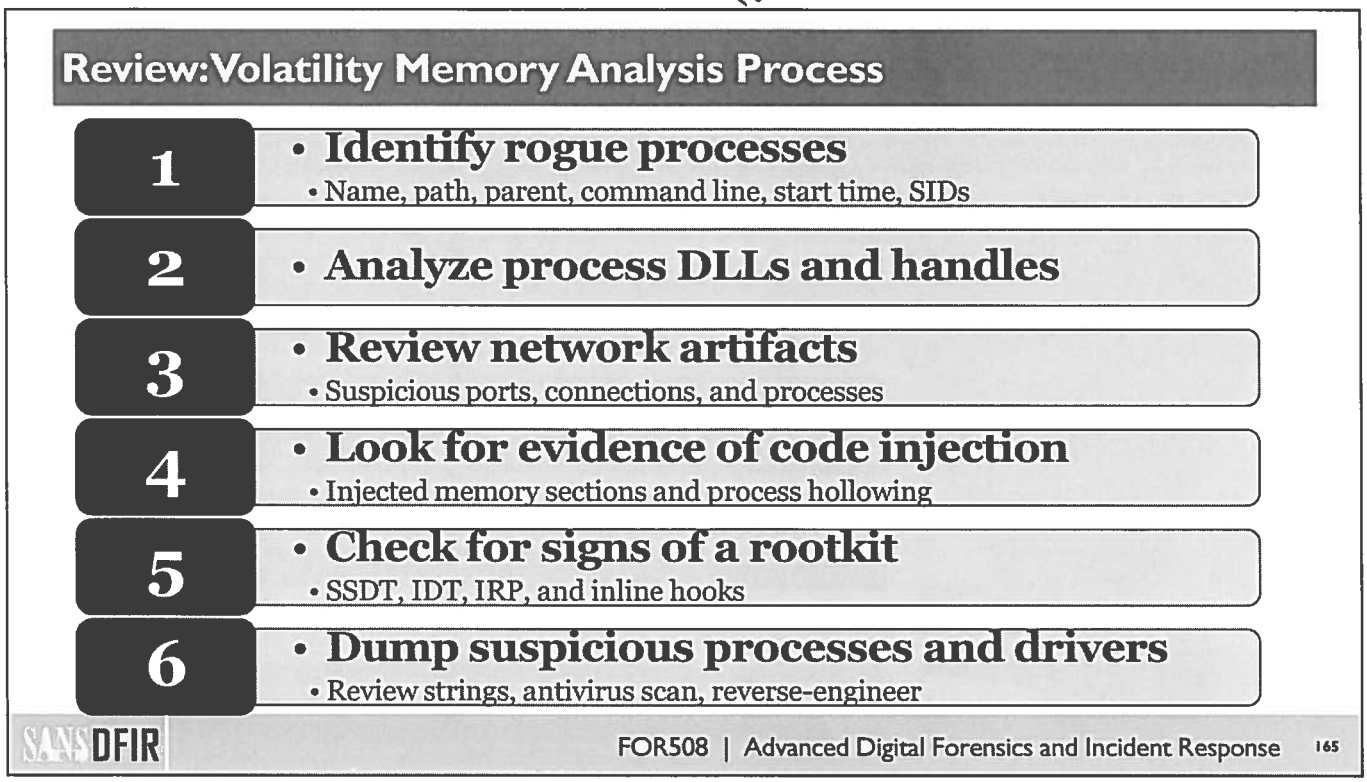


When you are down to approximately 10 to 20 candidates at this point, it is always best to identify where those files show up in your timeline and examine the temporal proximity of the file. The context that could be discovered here could quickly eliminate many more files from being actual malware based on the use in the timeline.

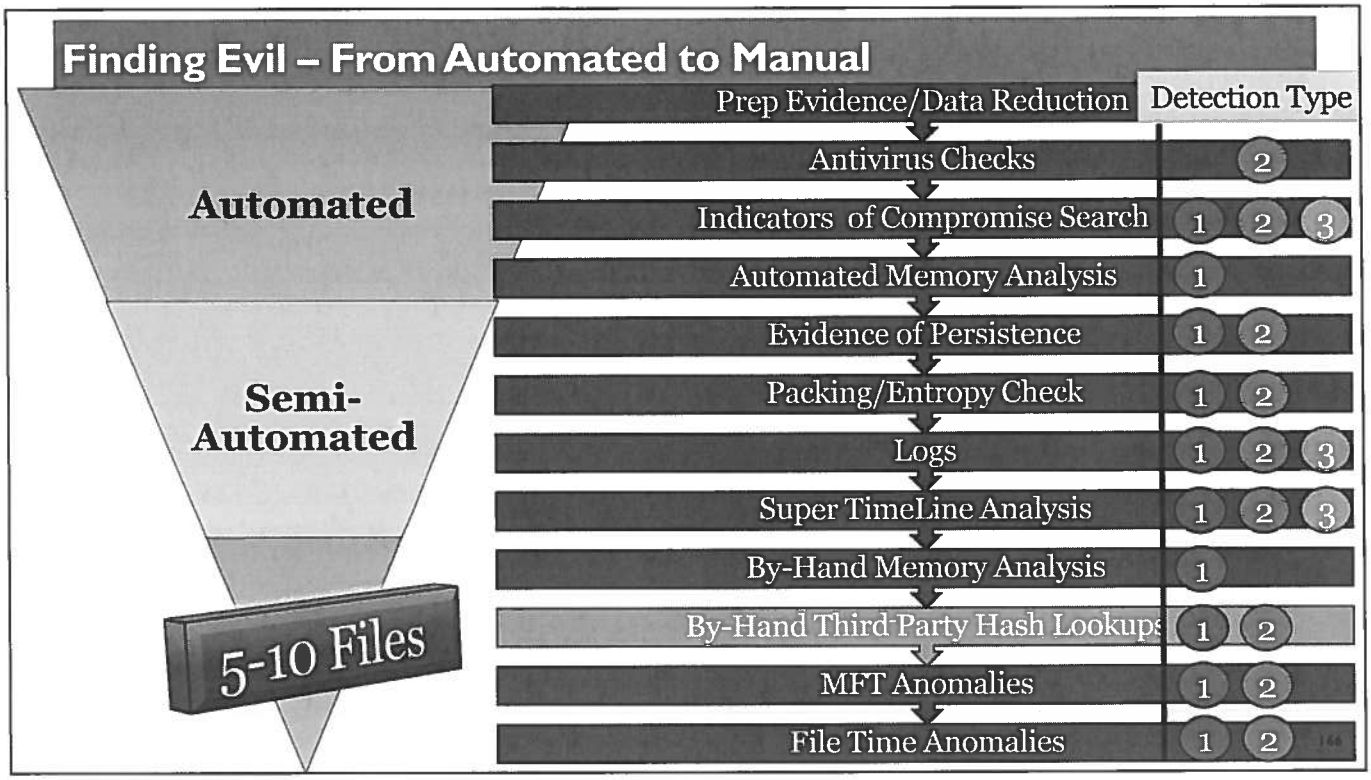
In contrast, in the above example, we can see the creation of the file winsvchost.exe in system32 directory. If this was one of your candidate files and you looked for when the file was created in the timeline, you would clearly see the traits that are seen during a **spearphishing** attack. Contextual clues in the temporal proximity of the files you are examining will be quite useful in your overall case.



This page intentionally left blank.

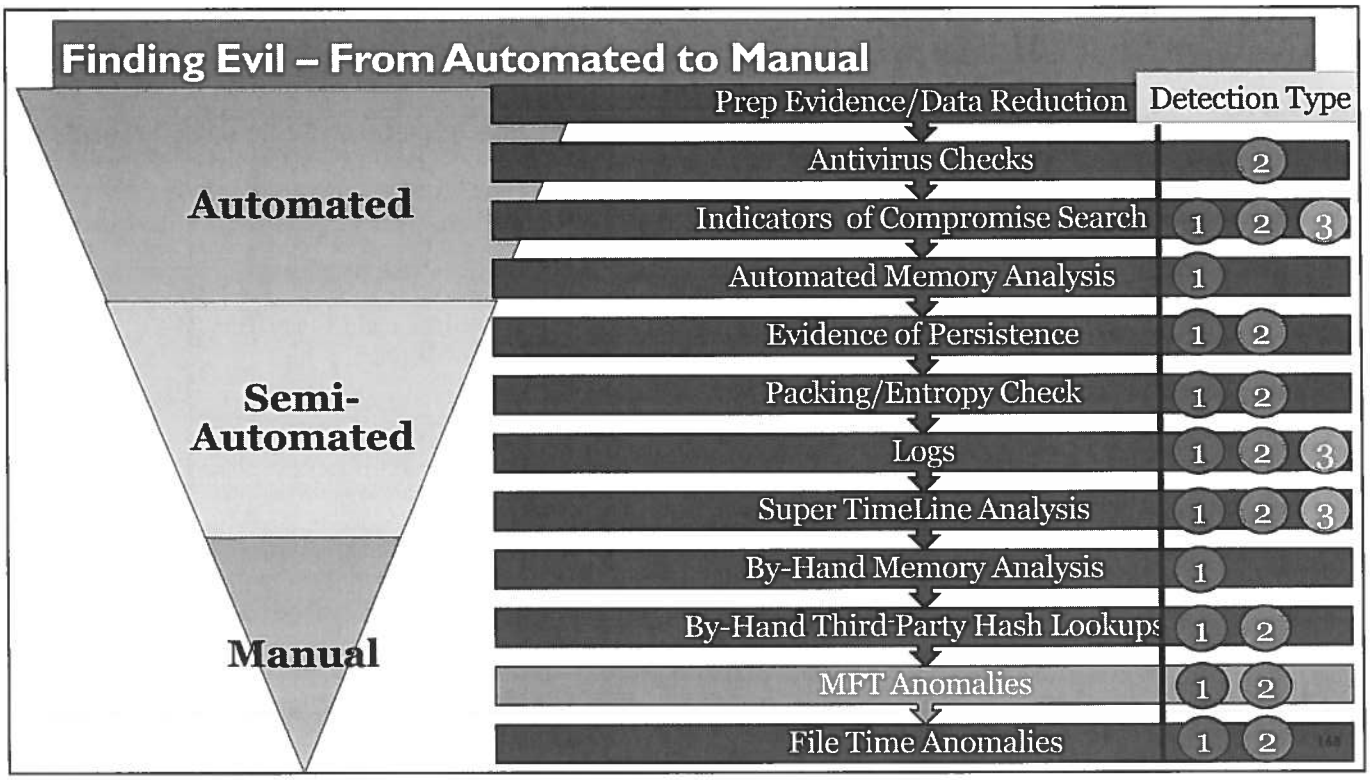


This page intentionally left blank.

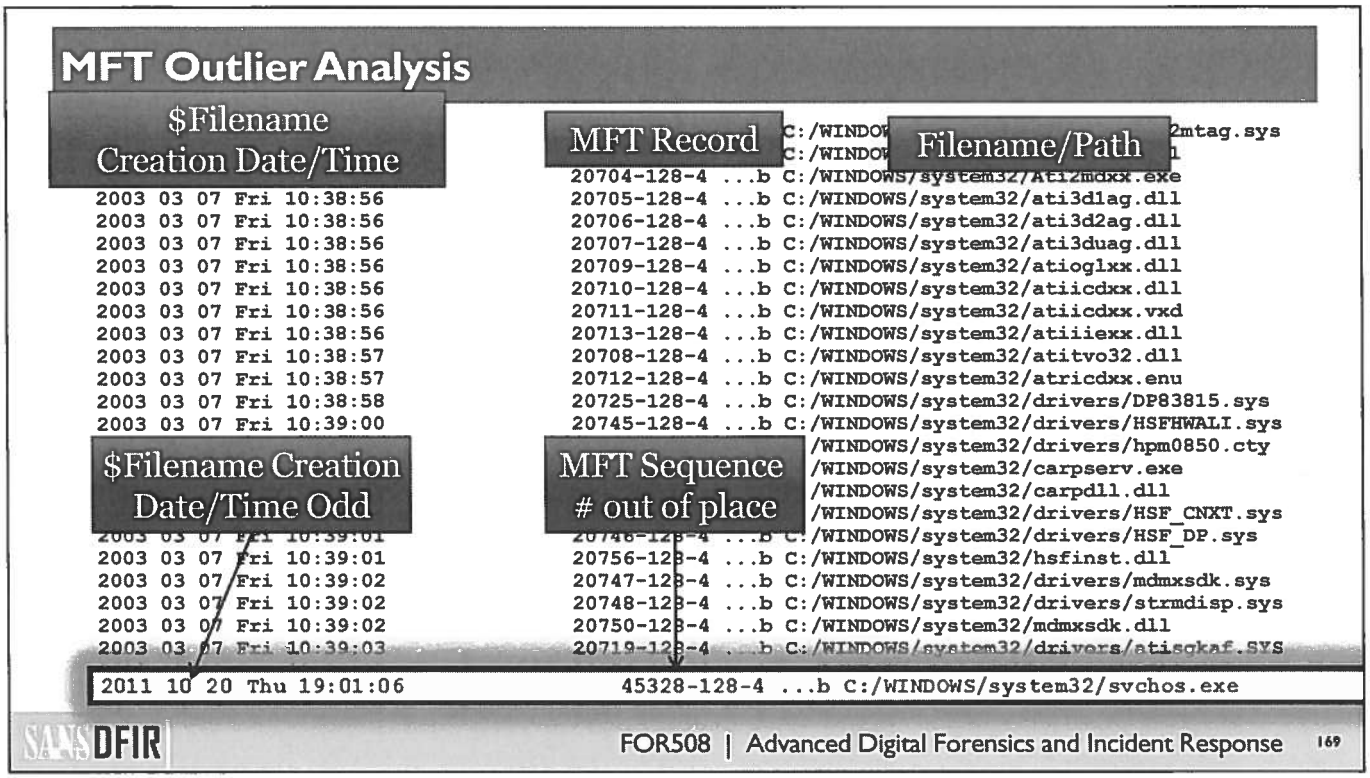


This page intentionally left blank.

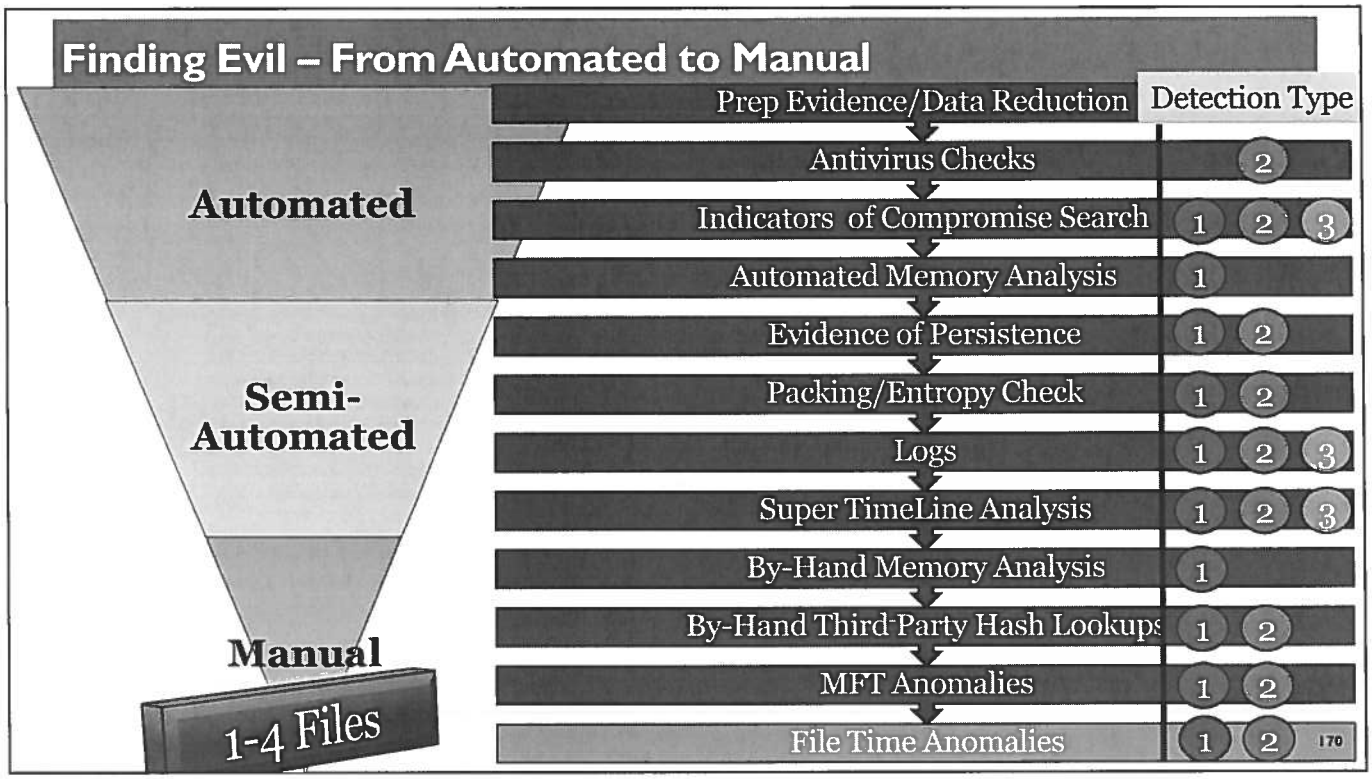
VirusTotal will scan a file through more than 40 different A/V scanners to determine if any of the current signatures detect the malware. VirusTotal can also receive MD5 hashes of the possible malware candidates. One of the downsides of using Virustotal by submitting a file is that you will lose control 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 responders opt to not submit their actual malware candidate files to the service as a result.



This page intentionally left blank.



This page intentionally left blank.



This page intentionally left blank.

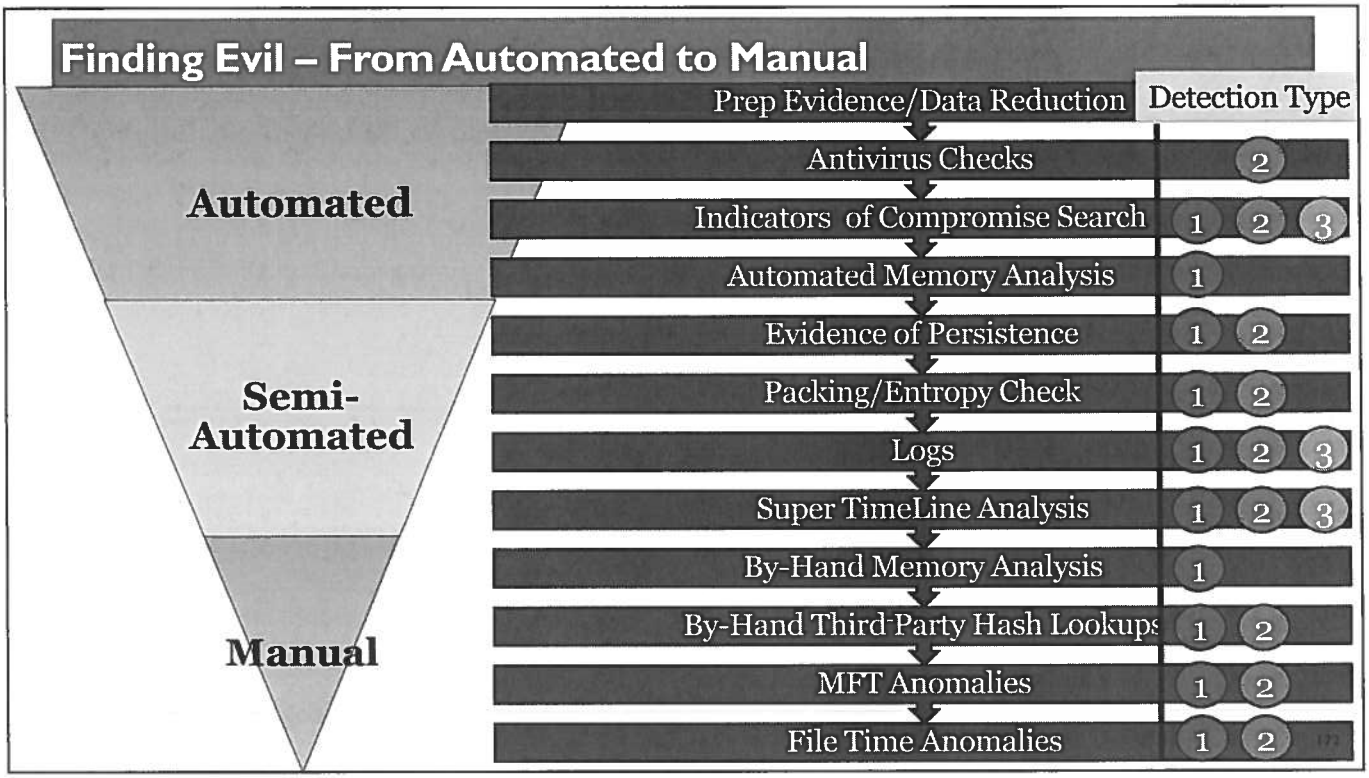
Timestamp Anomalies Hackers Are Crafty: We Are Better

- Timestamp Anomalies
  - $SI Time is before $FN Time
  - Nanosecond 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.

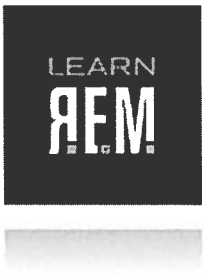
You can also compare this to the program compile time to see if the program were compiled after the creation date locally on the machine using the EXIFTool talked about in FOR408.



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.

Intrusion Forensic Challenge


SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

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

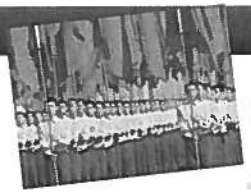
Exercise 6.1

Intrusion Forensic Challenge

This page intentionally left blank.

Course Review

The Threats



- **APT – Advanced Persistent Threat**
- **Organized Crime – Card Data Theft**
- **Hackivists – Expect Them**

The Reality

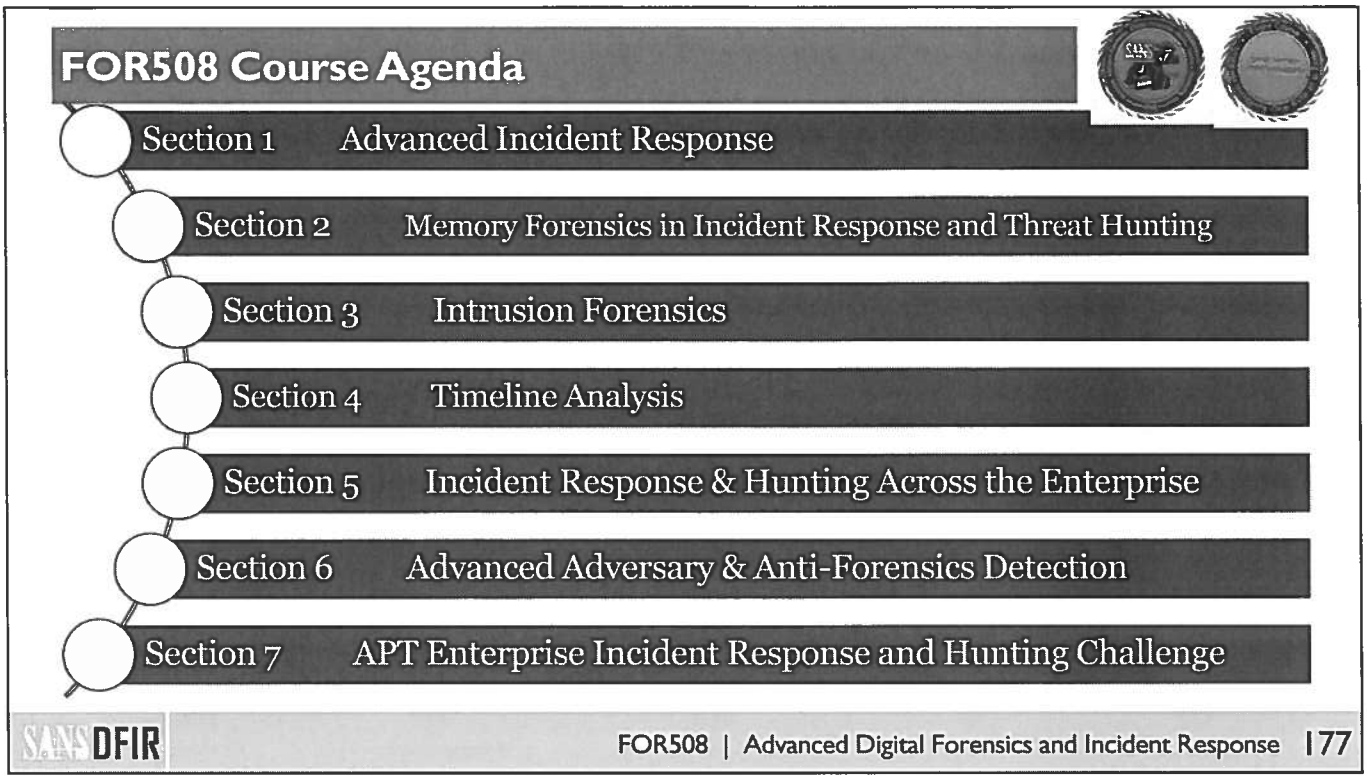
- Many organizations have a difficult time responding to intrusions from advanced adversaries

What you should learn by the end of the course

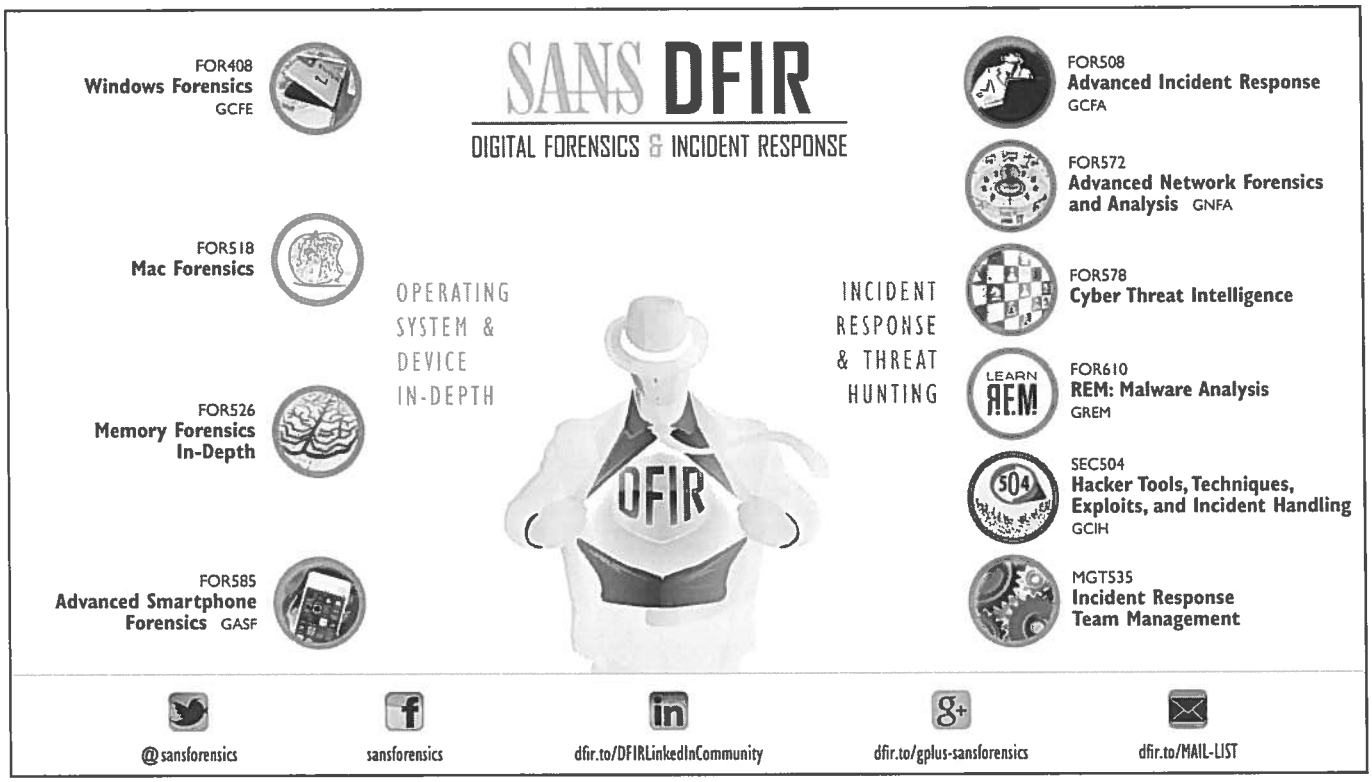
- Real Incident Response Tactics
- Timeline and SuperTimeline Analysis
- Memory Analysis
- Enterprise Investigations
- Anti-Forensic Detection
- Malware Detection
- IP Theft Detection

Lethal Forensicator

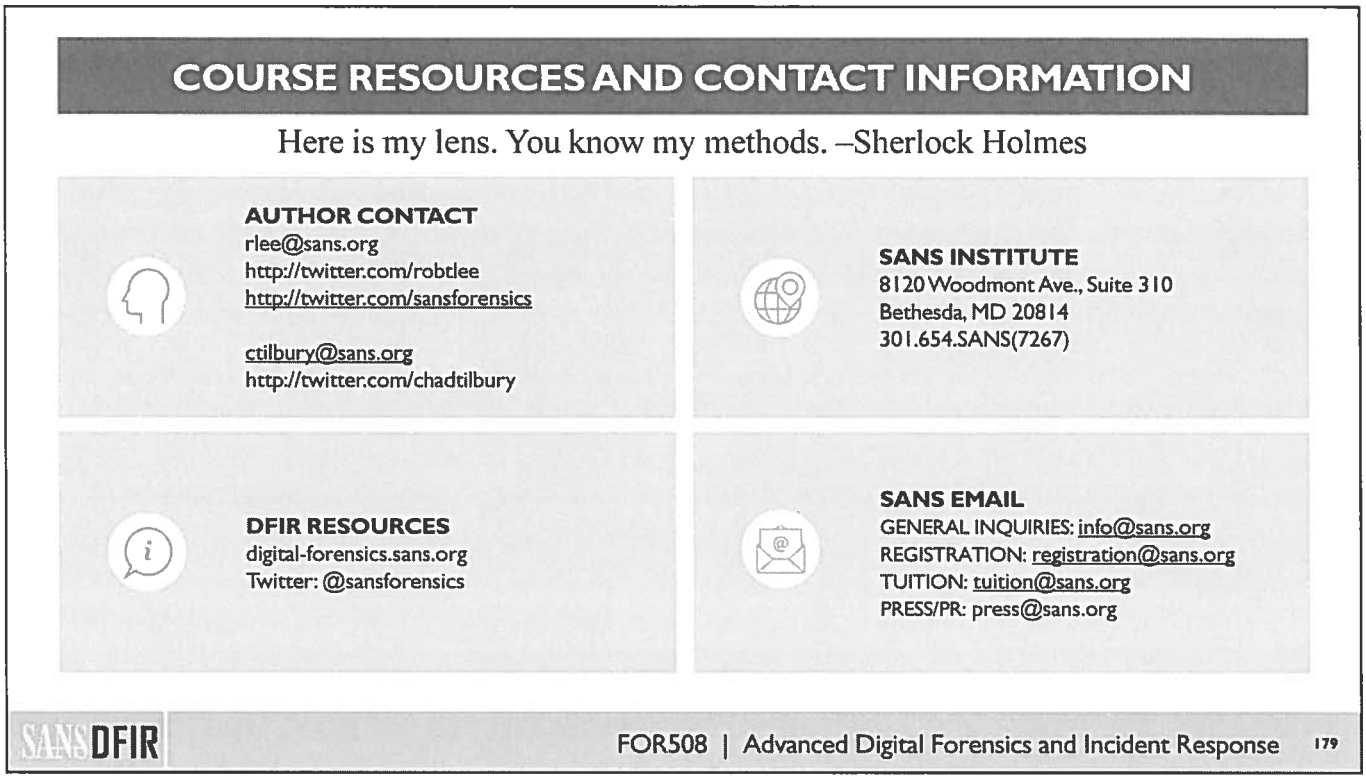
This page intentionally left blank.



This page intentionally left blank.



This page intentionally left blank.



This page intentionally left blank.