508.4

Timeline Analysis

All reference links are operational in the browser-based delivery of the electronic workbook.

FOR508.4 | Advanced Incident Response, Threat Hunting, and Digital Forensics

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

Timeline Analysis

Welcome to Section 4.

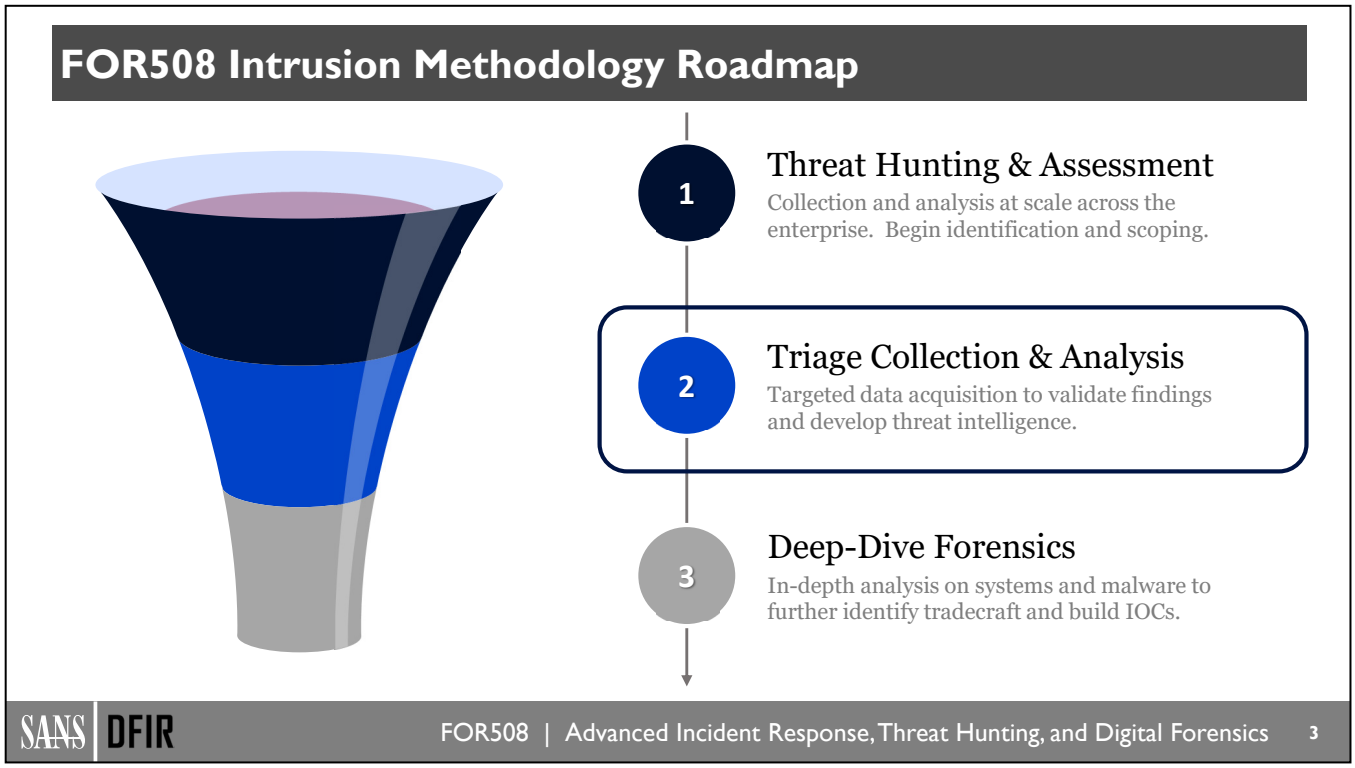
Author Team:

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

Chad Tilbury
ctilbury@sans.org
https://twitter.com/chadtilbury

Mike Pilkington
mpilkington@sans.org
https://twitter.com/mikepilkington

This page intentionally left blank.



Your journey through FOR508 has been designed to follow a standard workflow for performing threat hunting, compromise assessments, and incident response activities. The roadmap we will use in this class follows:

Threat Hunting & Assessment
We will start our process by looking at the network using tools that can scale collection and analysis, focusing on occurrence stacking and outlier analysis. Most attendees have thousands of endpoints necessitating broad scoping techniques at the start of an investigation.

Triage Collection & Analysis
As systems of interest are identified, we will perform targeted triage collection to acquire a deeper understanding of attacker activity. Triage data can include traditional forensic artifacts like application execution data, filesystem information, and in-memory artifacts such as process trees.

Deep-Dive Forensics
Finally, we will reserve our limited analyst time for performing deep-dive forensics on only a handful of systems having the best chance to help us understand attacker tools and tradecraft and craft better indicators to assist with scoping additional compromised systems.

Malware Discovery

This page intentionally left blank.

Malware Discovery: Anomaly Detection

YARA
- Search for string and header-based signatures
- Standard for IOC sharing
- Easy to create custom signatures to detect new tools/malware

Sigcheck
- Microsoft tool designed to validate digital signatures
- Create file hashes
- Virus Total support

DensityScout
- Checks for possible obfuscation and packing
- Files receive an entropy score
- Score can be used to identify whether a set of files merits further investigation

capa
- File capability identification
- Anti-analysis features?
- Contains an embedded .exe?
- Interesting host interaction?
- Code injection capabilities?

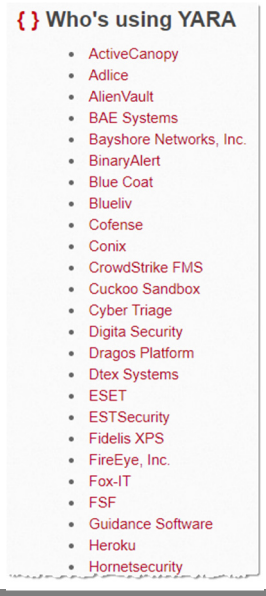
A common challenge facing incident responders is rapid identification of suspicious and malicious software on a computer system. There are hundreds of thousands of files and folders on a system and defenders can leverage a variety of techniques including file analysis, live response (the examination of a system's state while it's running) and temporal analysis (a fancy phrase for "Timelining") to identify suspicious files. On the flip side, attackers regularly employ data-hiding techniques to make identification even more difficult:

- 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/deobfuscation. For example, a "Packed Executable" is a piece of software containing an "unpacking" program and a payload.
- Custom data encryption/decryption routines

In this section we will cover several techniques designed to quickly characterize suspicious files. We will use YARA and Sigcheck for signature-based detection, DensityScout for file entropy analysis and capa to provide insight into suspicious capabilities of files of interest. Together these tools provide a powerful means to detect a wide-range of malware and anti-detection capabilities seen in the wild.

YARA Pattern Matching

- Open-source tool from VirusTotal designed to identify and classify malware
  - Large collection of available signatures and extensions
- Stand-alone tool is effective for quick scans of folders or suspected malware
  - Linux, Mac OS X, Windows
  - YARA is also baked into a wide-range of other tools



YARA is currently the most widely used indicator of compromise format. Its popularity stems from striking the right balance of simplicity and power, making it easy for malware analysts and incident responders to identify and classify malware samples. YARA rules are written to match patterns, and the rules themselves are easily understood by both machines and human operators. Originally designed by VirusTotal to help classify their enormous collection of submitted malware samples, it has been embraced by most major security vendors and YARA support is embedded in a vast number of tools.

The stand-alone YARA application can be run on Linux, Mac, and Windows and is fast and simple to use for a quick check of suspect files.[1] It takes a collection of signatures and target files as inputs, and outputs any signatures matches for those files.

[1] YARA - The pattern matching swiss knife for malware researchers: https://for508.com/378jh

Indicators of Compromise - YARA

```
rule SeaDuke_Sample {
      meta:
            description = "SeaDuke Malware"
            license = "https://creativecommons.org/licenses/by-nc/4.0/"
            author = "Florian Roth"
            reference = "http://goo.gl/MJ0c2M"
      strings:
            $s0 = "bpython27.dll" fullword ascii
            $s1 = "email.header(" fullword ascii
            $s2 = "LogonUI.exe" fullword wide
            $s3 = "Crypto.Cipher.AES(" fullword ascii
            $s4 = "mod is NULL - %s" fullword ascii
      condition:
            uint16(0) == 0x5a4d and filesize < 4000KB and all of them
}
```

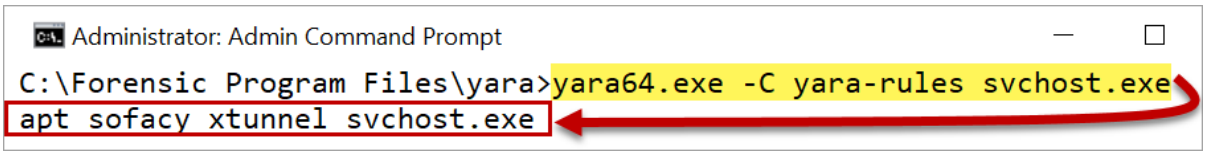
1 2 3

While many YARA rules are strings based, more sophisticated rules can be crafted using regular expressions, wildcards, conditions, and modules such as pe header components from the portable executable structures. In the example on this slide, the matched condition requires a 1) "MZ" portable executable signature, 2) a file size limit, and 3) matching of five specific strings found inside the file. The goal of an IOC is to create a signature that is specific enough to limit false positives at scale, while being broad enough to still match different variants of the same malware sample. This is a difficult balance and is why different IOCs have wildly different efficacy rates.

YARA Usage

```
yara64.exe -C compiled-rules-file <file or directory>
```

```
yara [options] RULES_FILE <file or directory>
   [Useful Options]
    -C:                  Load pre-compiled rules
    -c:                  Print only number of matches
    -f:                  Fast matching mode
    -w:                  Disable warnings
    -r:                  Recursively search directories
    -p <threads>:        Use specified number of threads during scanning
```



The YARA tool is a simple implementation of the standard, leaving more complicated implementations to other tools. It takes a single YARA rule file as input and scans a collection of files for signature matches (including the ability to search recursively). The rules file can be frustrating for first time users because there is no capability to point the tool at a folder of collected signatures. If you are using multiple signatures, they must all be appended into the same signature file or referenced as part of an "index" file, as seen below. A rules file can be compiled using the yarac64.exe compilation tool, and then referenced using the "-C" option on the YARA command line. Compiling signatures is a good best practice for larger signature sets because it significantly increases the efficiency of the scanning engine. The example on this slide uses the Windows (.exe) 64-bit version of YARA with a pre-compiled set of rules named "yara-rules". One hit was returned, with the file named "svchost.exe" matching a signature for "apt sofacy xtunnel". The Yara-Rules GitHub site is an excellent starting place for building a good rule set to use for malware scanning.[1]

Example Contents of a YARA Index File:

include ".\exploit_kits\EK_ZeroAcces.yar"
include ".\exploit_kits\EK_Zerox88.yar"
include ".\exploit_kits\EK_Zeus.yar"
include ".\malware\000_common_rules.yar"
include ".\malware\APT_APT1.yar"
include ".\malware\APT_APT10.yar"
include ".\malware\APT_APT15.yar"
include ".\malware\APT_APT17.yar"
include ".\malware\APT_APT29_Grizzly_Steppe.yar"

[1] GitHub Yara-Rules: https://for508.com/d8tvr

DensityScout: Entropy/Packing Analysis

```
densityscout -pe -r -p 0.1 -o results.txt <directory-of-exe>
```

```
densityscout [options] file or directory
   [Useful Options]
    -a:                 Show errors and empties, too
    -d:                 Just output data
    -l:                 Lower than the given density
    -n:                 Print number lines
    -m:                 Mode ABS (default) or CHI (for filesize > 100 Kb)
    -o file:            File to write output to
    -p density:         Immediately print if lower than the given density
    -r:                 Walk recursively
    -s suffix(es):      Filetype(s) (i.e., dll or exe,...)
    -S suffix(es):      Filetype(s) to ignore (i.e., dll or dll,exe)
    -pe:                Include all portable executables by magic number
    -PE:                Ignore all portable executables by magic number
```

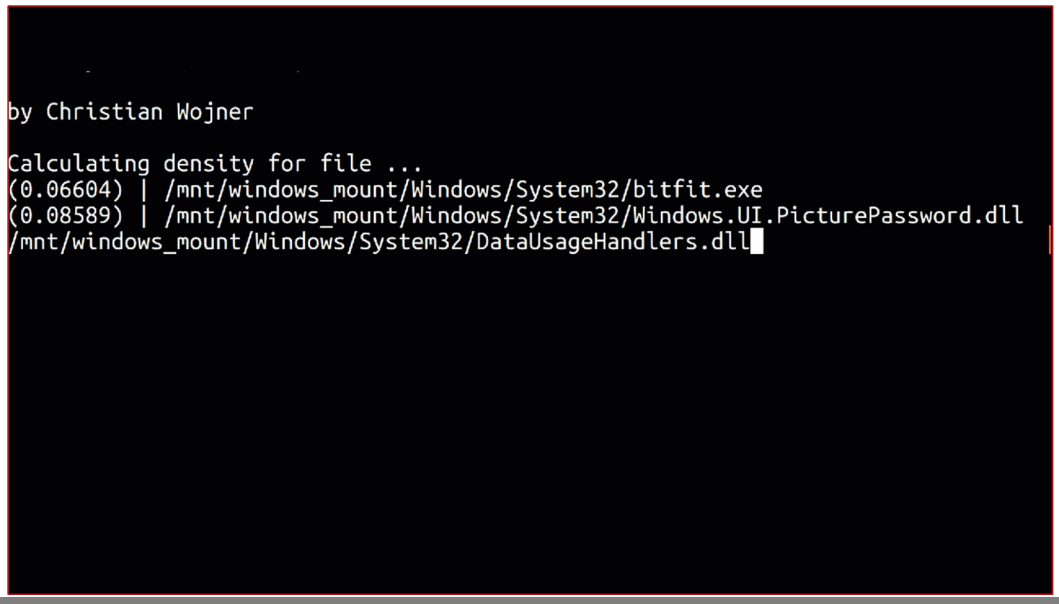

DensityScout is a tool written for one purpose: finding suspicious files. Written by Christian Wojner at CERT Austria, it attempts to detect common obfuscation techniques like runtime packing and encryption.[1] DensityScout scans a desired file system, calculating the density of each file. What exactly is density? In simple terms, density is a measure of randomness, or entropy, of a file. Files that are encrypted, compressed, or packed have a large amount of inherent randomness, making them different from normal files. Legitimate executables in Windows are rarely packed or encrypted and hence rarely exhibit random character sequences leading to higher entropy. Of the tens of thousands of Microsoft .exe, .dll, and .sys files on a system, almost none will exhibit significant entropy. Why does this tool call its output "density"? Christian decided to not use the well-known word "entropy" for the mathematical concept to circumvent any tiresome philosophical discussions with mathematicians with very narrow definitions of the term.

```
densityscout -pe -r -p 0.1 -o results.txt c:\Windows\System32
```

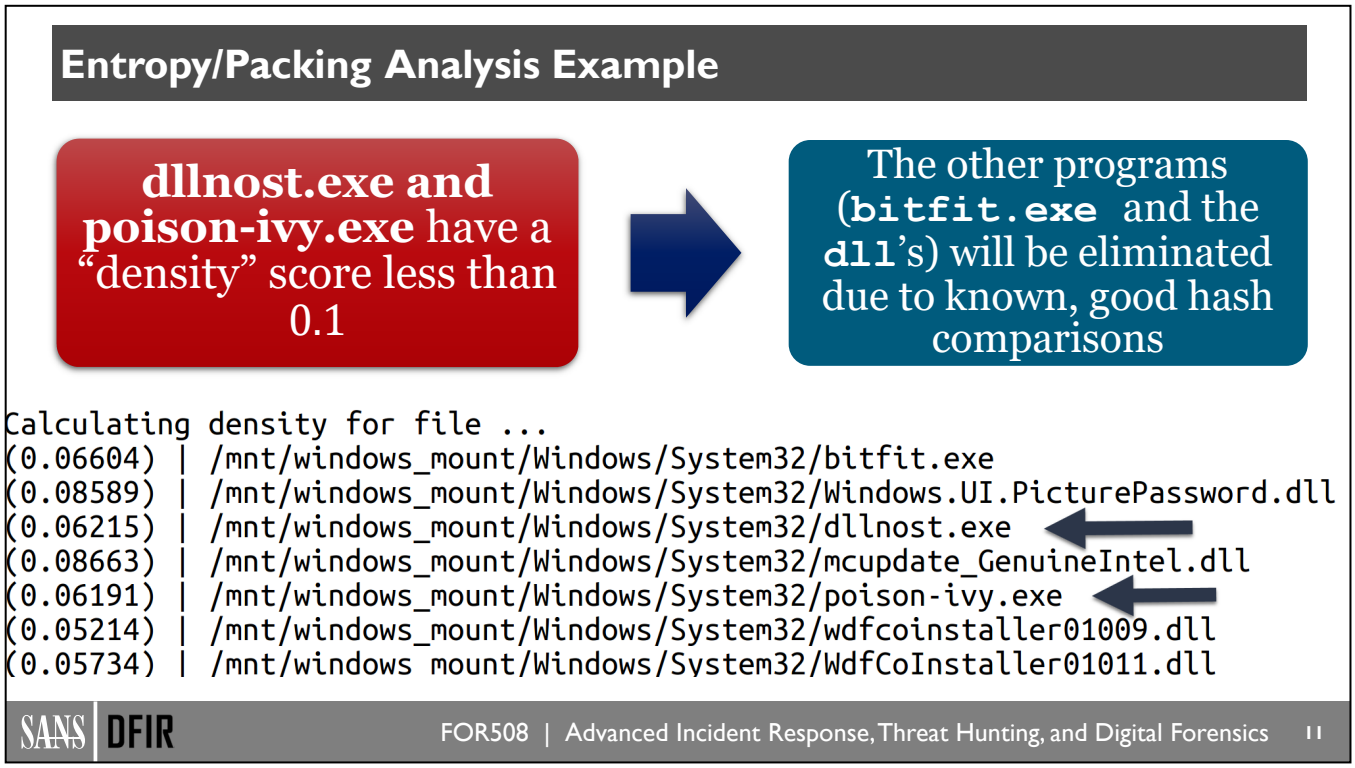
The "-pe" option tells DensityScout to select files by using the well-known signature of portable executables ("MZ"), instead of just selecting for files by extension. This can help identify executable files which have been renamed to hide in plain sight. "-r" instructs the tool to scan all files and sub-folders recursively from the starting point. The next option, "-p 0.1" is for the impatient. With this option, you can instruct DensityScout to display to screen each file found with a density below the provided threshold. If you do not use this option, you must wait until DensityScout is finished and outputs a descending list of all results. The value "0.1" in the example is a good threshold to use to keep findings visually manageable. The smaller the value, the denser, or randomer, the file is. The typical density for a packed file is less than 0.1, whereas the typical density for a normal file is often greater than 0.9. Finally, the option "-o results.txt" is the output file. This file will contain the density value for every evaluated file.

[1] DensityScout from CERT.at: https://for508.com/ltnck

Example DensityScout Execution



In this example, DensityScout was executed with the "-p 0.1" option, telling it to print to screen any files exhibiting large amounts of randomness.



In the above example, DensityScout was executed against the C:\Windows\System32 directory of a potentially compromised system. A total of seven files were identified as highly entropic (out of thousands). Of this group, two turned out to be malware: dllnost.exe and poison-ivy.exe. Of course, all seven files had to be investigated, and the benign files were eliminated via hash comparisons with a known-good database and by reviewing a baseline image of a system known to not be compromised. An important concept throughout this section is one of data reduction. While no tool will ever have a 100% hit rate for malicious software, the goal is to reduce the number of files that must be looked at to a manageable set.

Digital Signature Checking `sigcheck.exe`

- Verify that images are digitally signed and dump version information with this simple command line utility

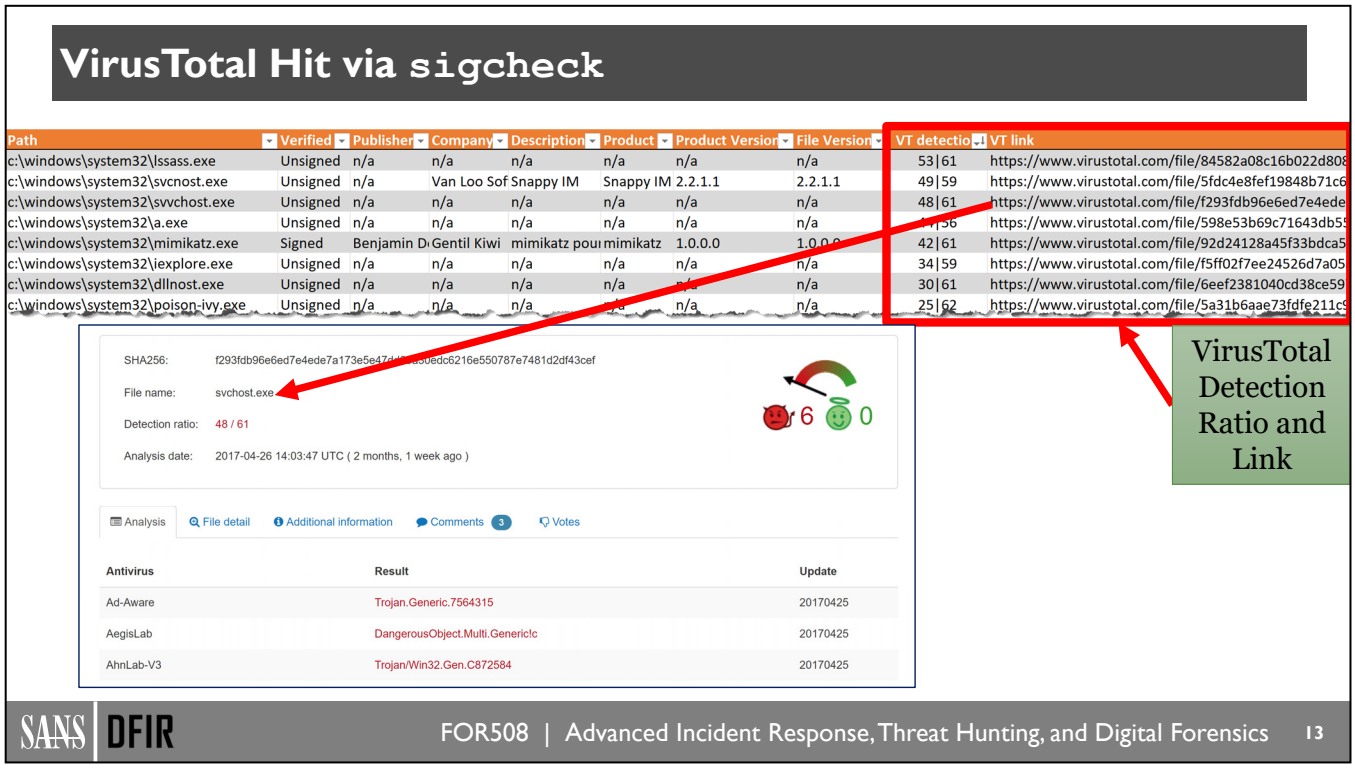
```
C:\> sigcheck –c –e –h –v <dir-of-exe> >  sigcheck-results.csv
```

```
sigcheck [options] file or directory
    [Useful Options]
      -a                Show extended version information
      -c                csv output
      -e                Scan executable images only
      -h                Show file hashes
      -s                Recurse subdirectories
      -u                Show files that are unknown if VirusTotal check
                        is enabled; otherwise, show only unsigned files.
      -v[rs]            Query VirusTotal for malware based on file hash. Add 'r' to
                        open reports for files with non-zero detection. Files reported
                        as not previously scanned will be uploaded to VirusTotal if the
                        's' option is specified.
      -t[u][v]          Dump contents of specified certificate store ('*' for all
                        stores). Specify -tu to query the user store (machine
                        store is the default).  Append '-v' to have Sigcheck download
                        the trusted Microsoft root certificate list and only output
                        valid certificates not rooted to a certificate on that list. If
                        the site is not accessible, authrootstl.cab or authroot.stl in
                        the current directory are used instead, if present.
```


Sigcheck is a favorite tool from the free Sysinternals toolkit. The simple purpose of Sigcheck is checking the code signing characteristics of executable files in a directory. By using several of the options, namely, -u and -s, an analyst can quickly dump a recursive list of unsigned binaries for analysis.

The vast majority of malware in the wild is not signed. This is good news, because with the release of Windows 64-bit operating systems, the majority of legitimate code is signed. Those two facts combine to provide a powerful detection capability! There are examples of advanced malware with valid digital signatures (valid at least at the time of discovery); Flame and Stuxnet are two commonly cited examples. Sometimes signing malware is worth the effort to a specific adversary to improve the chances of blending in, but there is a cost. Once the malware is detected and the code certificate is revoked, it could reveal the malware in other locations. It is very much 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 after the code signing certificate has been revoked. This keeps the number of signed malware samples down and increases the effectiveness of looking for unsigned code on a system, a task at which Sigcheck excels.

Systinternals provides regular updates to its toolkit, providing new capabilities to keep pace with new attacks. Integration with VirusTotal to perform hash lookups was recently added. Sigcheck has also added the capability to dump certificate stores to identify any root certificates present not explicitly trusted by Microsoft. While outside the scope of this class, this capability could be a very useful way to identify a cloned and trusted root certificate.



Sigcheck does an excellent job of identifying unsigned executables. It also nicely looks up the hash of the executable in VirusTotal for you, allowing you to scan each static executable through many anti-virus engines online. If a hash results in a detection, the VirusTotal link is provided as output in the final column.

Enumerate Malware Capabilities: capa

- Triage detection using crowdsourced code patterns (rules)
  - File Header
  - API Calls
  - Strings and Constants
  - Disassembly Features
- Rules match common malware actions
  - Communication, Host Interaction, Persistence, Anti-Analysis, etc.
  - ATT&CK technique mapping also included
- Designed to provide capabilities in plain language to speed-up investigations

```
rule:
 meta:
   name: create reverse shell
   namespace: c2/shell
   author: moritz.raabe@fireeye.com
 features:
   - or:
     - and:
       - match: create pipe
       - api: kernel32.PeekNamedPipe
       - api: kernel32.CreateProcess
       - api: kernel32.ReadFile
       - api: kernel32.WriteFile
     - and:
       - match: create process
       - match: read pipe
       - match: write pipe
     - and:
       - match: create pipe
       - match: create process
       - basic block:
         - and:
           - count(api(SetHandleInformation)):
             2 or more
           - number: 1 = HANDLE_FLAG_INHERIT
```


The FireEye FLARE team released the open-source capa project in an attempt to lower the bar for initial malware investigations.[1] They provided the following problem statement: "Effective analysts can quickly understand and prioritize unknown files in investigations. However, determining if a program is malicious, the role it plays during an attack, and its potential capabilities requires at least basic malware analysis skills. And often, it takes an experienced reverse engineer to recover a file's complete functionality and guess at the author's intent."[2] Reverse engineering capabilities are in short supply on nearly every security team and thus it is extremely helpful for incident responders to be able to perform "field triage" activities to identify a suspicious file as malicious and to understand how it might have been used in an attack. The capa tool provides more information to responders than ever before by actually disassembling the code, looking for well-known patterns, and then describing what those patterns indicate in plain language. Imagine convening some of the best reverse engineers on the planet and having them brainstorm on how they most commonly identify the capabilities of a piece of software. Capa accomplished this by building an easy mechanism for defining rules and then crowdsourcing rule creation! The FireEye FLARE team seeded capa with hundreds of patterns and researchers have added many more. The end result is a tool anyone can use that takes advantage of the "hive-mind" knowledge of the reverse engineering community. Not only that, but the rules are written in YAML and open-source, providing an exciting opportunity to explore these patterns and learn about the underlying techniques. As an example, the rule on this slide is looking for reverse shell communication capabilities and you can clearly see that (named) pipes are a primary flag. This pattern would perhaps be useful in identifying command and control techniques leveraging named pipes like those used by the Cobalt Strike Beacon.

[1] GitHub - fireeye/capa: https://for508.com/boxkq
[2] Automatically Identify Malware Capabilities: https://for508.com/n8skp

capa Usage

```
capa.exe -f pe -v <file>
```

```
capa [options] <file>
   [Useful Options]
    -v:                 Enable verbose results
    -vv:                Enable very verbose results
    -f <format>:        Choose between PE or shellcode analysis (pe,sc32,sc64)
    -r RULES:           Specify alternative rules directory
    -t TAG:             Filter on a specific rule meta field value
    -j:                 Output JSON instead of text
```

** The **capa** project includes compiled versions for Windows, Mac, and Linux and can alternatively be run natively in Python

Capa is designed to be run against a single file of interest. In its simplest form you can point it at an executable with no options and it will present a report of the capabilities of that sample. If you would like more information after seeing the initial report, run the tool again with the verbose options ("-v" or "-vv") to identify exactly where matches were found in the code. This "verbose" information is likely most interesting to reverse engineers who plan to visit those locations using a disassembler tool, but it can give more context even to a lay investigator. An important option to understand is the format (-f) command. By default, capa will assume you are asking it to analyze an executable ("-f pe"), but capa can also parse shellcode! This can be particularly useful for analysis of injected memory sections that were dumped with tools like Volatility malfind, or shellcode pulled out of encoded PowerShell scripts (PowerShell Empire and Cobalt Strike are fond of this technique). It is very likely that you will not know whether the shellcode is 32-bit (format option "sc32") or 64-bit (format option "sc64"), so feel free to try both and see which option provides results. Finally, if you are automating analysis, you might prefer output in JSON format, which can be easily parsed for further interaction.

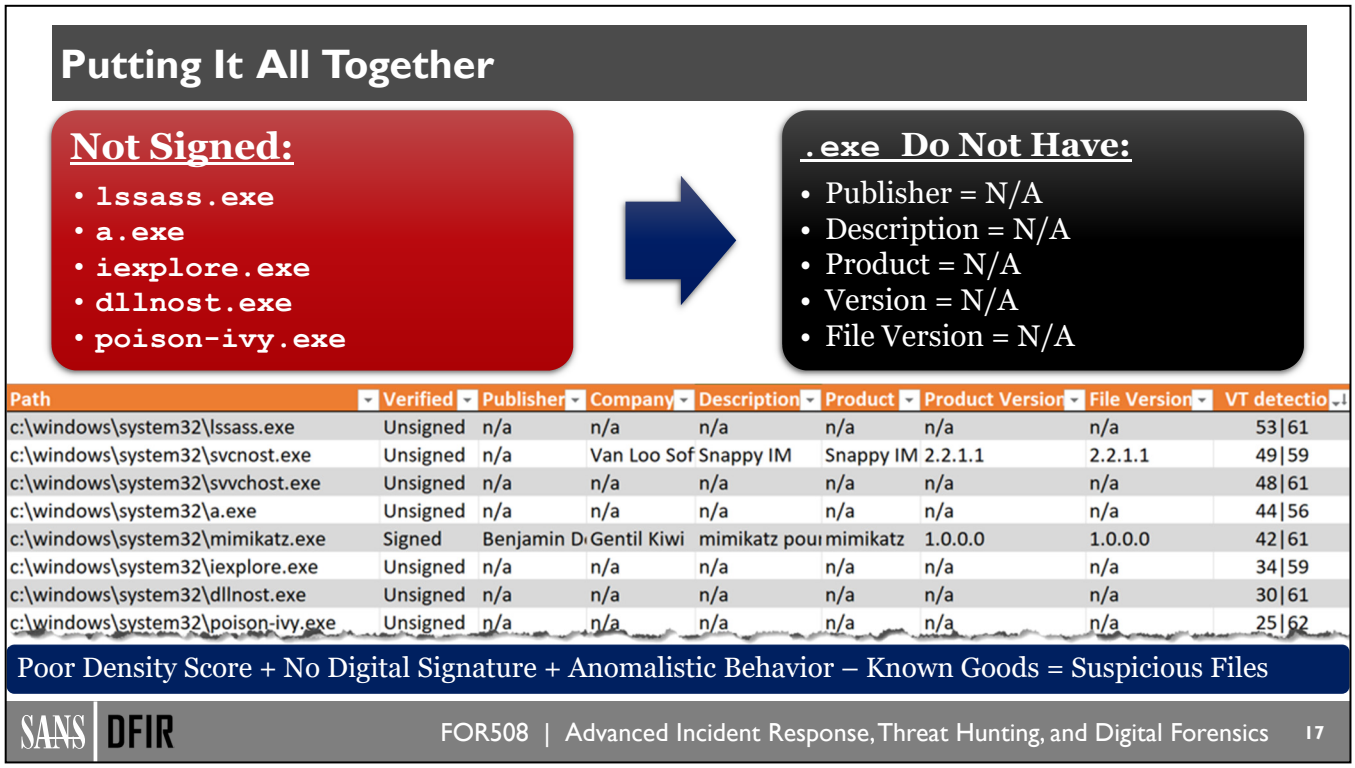
capa Analysis Example


```
# ./capa poison-ivy.exe
loading : 100%|[redacted]| 469/469 [00:00<00:00, 1979.08     rules/s]
matching: 100%|[redacted]| 1/1 [00:00<00:00,  9.06 functions/s]
+------------------------+-------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                    |
|------------------------+-------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Virtualization/Sandbox Evasion::System Checks [T1497.001]                           |
+------------------------+-------------------------------------------------------------------------------------+
+----------------------------+-------------------------------------------------------------------------------+
| MBC Objective              | MBC Behavior                                                                  |
|----------------------------+-------------------------------------------------------------------------------|
| ANTI-BEHAVIORAL ANALYSIS   | Debugger Detection::Process Environment Block BeingDebugged [B0001.035]       |
|                            | Virtual Machine Detection::Instruction Testing [B0009.029]                    |
|                            | Virtual Machine Detection [B0009]                                             |
| EXECUTION                  | Install Additional Program [B0023]                                            |
+----------------------------+-------------------------------------------------------------------------------+
+-----------------------------------------------------+-----------------------------------------------------+
| CAPABILITY                                          | NAMESPACE                                           |
|-----------------------------------------------------+-----------------------------------------------------|
| check for PEB BeingDebugged flag                    | anti-analysis/anti-debugging/debugger-detection     |
| execute anti-VM instructions                        | anti-analysis/anti-vm/vm-detection                  |
| reference anti-VM strings targeting Xen             | anti-analysis/anti-vm/vm-detection                  |
| compiled with Borland Delphi                        | compiler/delphi                                     |
| contain a resource (.rsrc) section                  | executable/pe/section/rsrc                          |
| contain a thread local storage (.tls) section       | executable/pe/section/tls                           |
| contain an embedded PE file                         | executable/subfile/pe                               |
```


The capa output is designed to be understandable and allow for further research into stated capabilities. At the bottom of the results will be a list of probable capabilities (CAPABILITY) for the sample analyzed along with the category of rule they matched (NAMESPACE). This is a good place to start. Above this information will be the Mitre ATT&CK and Malware Behavior Catalog (MBC) references for the identified capabilities, providing places for more research.[1] Keep in mind that legitimate software also has plenty of capabilities! Imagine running capa on a legitimate copy of powershell.exe. capa would tell you that PowerShell has the capability to interact with the registry, terminate processes, read and write files, etc. You can see how it would be easy to jump to conclusions. Thus, it is important to pair this information with other suspicious data points that led you to the file like it not having a valid digital signature or having highly entropic characteristics. A malicious file will typically fail multiple different checks on its legitimacy.

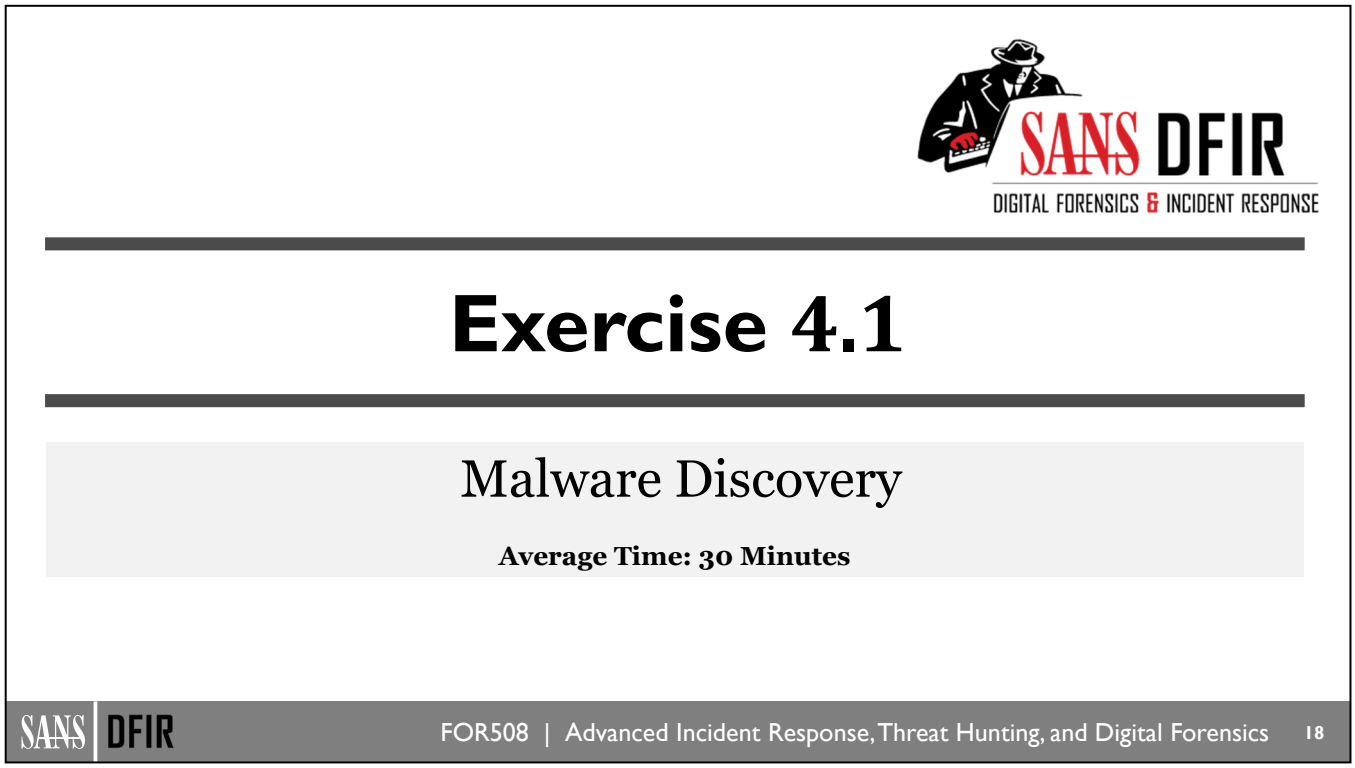
In the example on this slide, we see capa analyzing the ever-popular Poison Ivy remote access tool. The anti-debugging/analysis characteristics of this sample are a good indication of malicious intent as they do not appear very commonly in legitimate software. The same goes for the "contain an embedded PE file" and "compiled with Borland Delphi" capabilities. Gaining more experience with malware characteristics can help you better utilize a tool like capa. So, feel free to run it on the variety of samples we come across in this class and those you collect in your own investigations!

[1] GitHub – MBCProject: https://for508.com/2w3sa



In this example, many of the files in red are suspicious, having a low-density score, no digital signature and/or demonstrating suspicious file capabilities. Further, you might imagine we also checked the hashes of these files to see if they match with any known good software. The files failing these checks are the true outliers and would be worth further examination.

As you get experience with the output from the tools discussed here, you will learn to spot common omissions in a lot of malware. Malware authors often do not add publisher, description, product, version, or file version information to their creations. Malware by definition is anomalous and coded by bad actors with a wide range of skills, knowledge, and time. Malware rarely exhibits good coding practices found in a formal software development shop. Sadly, malware is not alone here, as there is plenty of poorly coded valid software. However, by analyzing files in multiple different ways, we can get a small enough set of results to increase our chances of finding malicious and interesting files.



This page intentionally left blank.

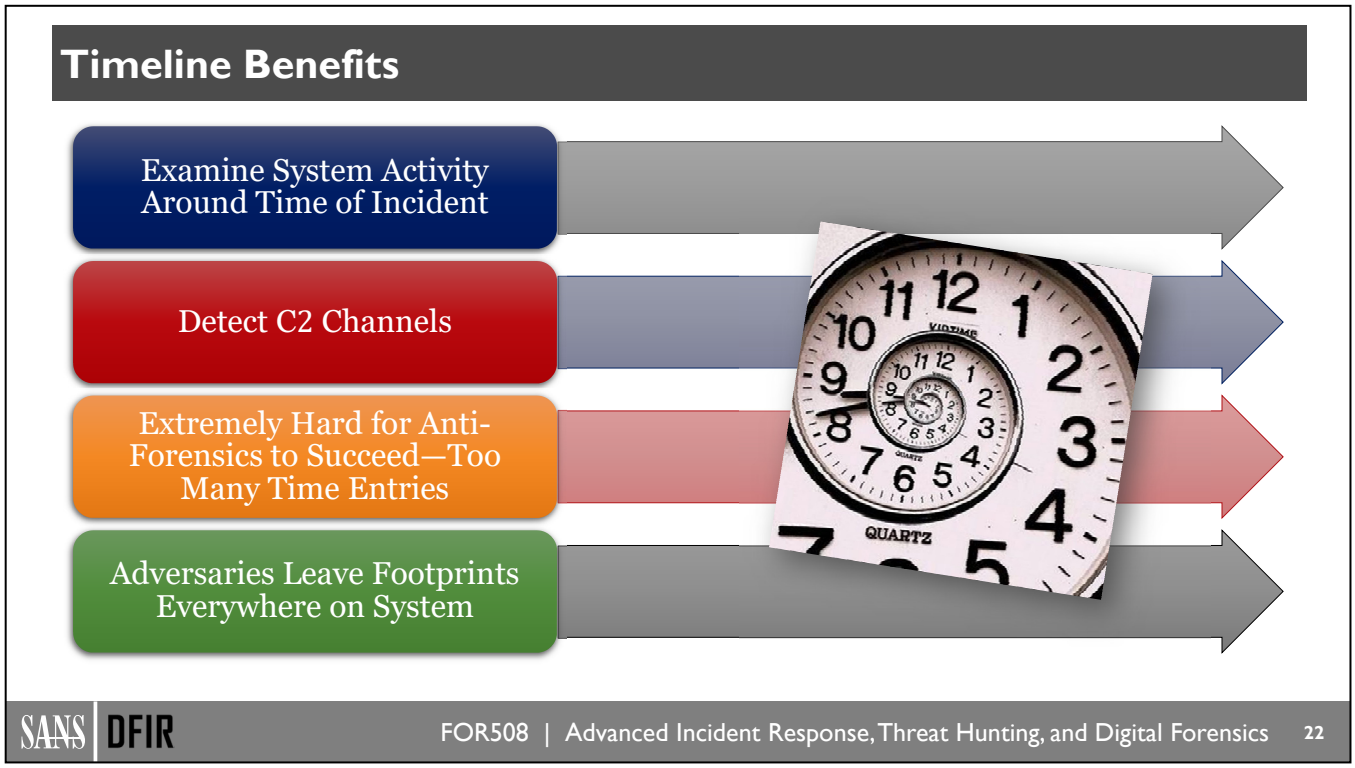
Timeline Analysis

This page intentionally left blank.

This page intentionally left blank.

Timeline Analysis Overview

This page intentionally left blank.



Timeline analysis has grown to be one of the more powerful techniques to speed up analytical work in digital forensics and incident response. It allows an investigator to automatically assemble artifacts from the registry, filesystem, and operating system in temporal order via a single tool. The power of timelining is seeing the interaction of artifacts and the result is a sequence of events telling a story of what happened on a system during a specific time frame. However, unlocking the power requires a skilled analyst able to infer the meaning of groups of artifacts. With practice, timeline analysis gives an investigator the best possible method to quickly assemble the pieces of a puzzle and answer important forensic questions.

Timeline data can be extremely beneficial when tracking adversary activity during intrusions. Because data is sourced from the operating and filesystem and not the network level, encryption and even covert tunnelling are irrelevant. It doesn't matter if the network level is encrypted, as the user must eventually interact with the system. Anything a user accomplishes will interact with the system somehow (opening files, starting programs, creating a socket, deleting files, or even anti-forensics) leaving behind artifacts to analyze .

Occasionally you may hear it stated that timelines are not to be trusted due to anti-forensics. This is a narrow view of how much time-based data actually exists on an operating system. It is nearly impossible for a determined hacker to delete all their footprints on the system. There are just too many. In addition, even if hackers have access, they usually have to bring their anti-forensics tools with them. Moving tools to a machine to delete data that leaves a new trail to detect—the anti-forensic tools themselves. As a result, try to think through this riddle. If a hacker uses a file wiper to wipe files what will the hacker use to wipe the wiper's existence?

Timeline Utopia

When you start your journey with timeline analysis, it sounds like it will be as easy as tracking footprints in the snow. It seems that it could be the answer to every forensic problem analysts have faced over the past decade of performing digital forensics. While timelines can answer many questions, the reality of timeline analysis is far different.

Even though you may be tracking a single user interacting with a machine, you might have multiple users to contend with. You also need to separate the frequent background system process and account interactions (e.g., System, Local Service, and Network Service). You might also need to contend with remote users and legitimate administrative activity. While "timeline utopia" does occur, the reality is often far messier.

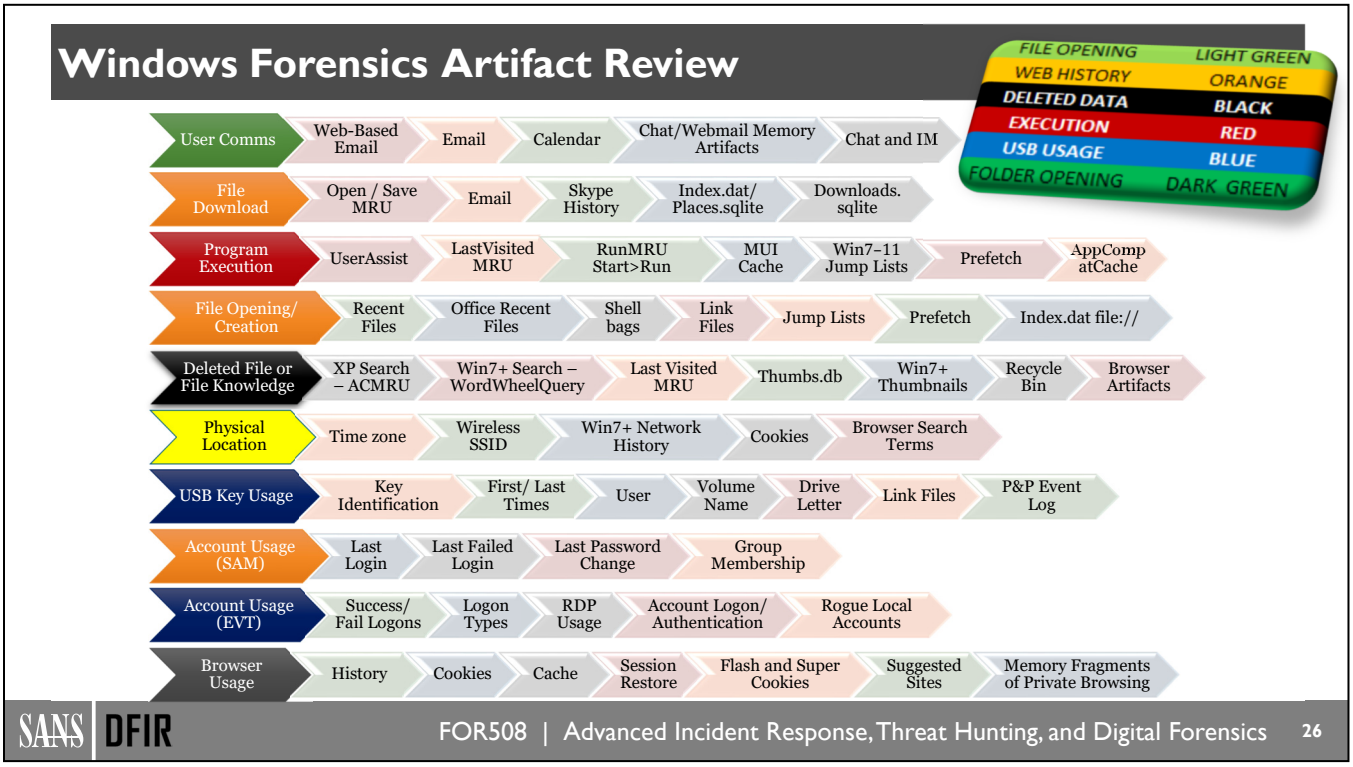
Timeline Analysis: The Reality

Most systems have multiple internal accounts, processes, and more creating constant time-related events. This "noise" makes tracking a specific series of events much more difficult. However, there's hope. Most activity will be independent of one another. Think of it as listening to three different types of music at once, classical, jazz, and rock. The combination might sound like a cacophony of noise, but with some effort you learn to pick out the classical threads. Similarly, we will be picking out the threads of activity related to the account or person we are investigating. A wealth of activity is tied directly to user accounts. It is unlikely the "SYSTEM" user itself will suddenly pop open a browser and start chatting with friends on Facebook. We may be able to determine interactive sessions versus remote sessions. The reality of examining many different transactions caused by multiple users, internal processes, automated tasks, and uninvited guests is a challenge, but not an insurmountable one. This section will teach the skills necessary to analyze timelines effectively.

The Windows Forensic Trinity

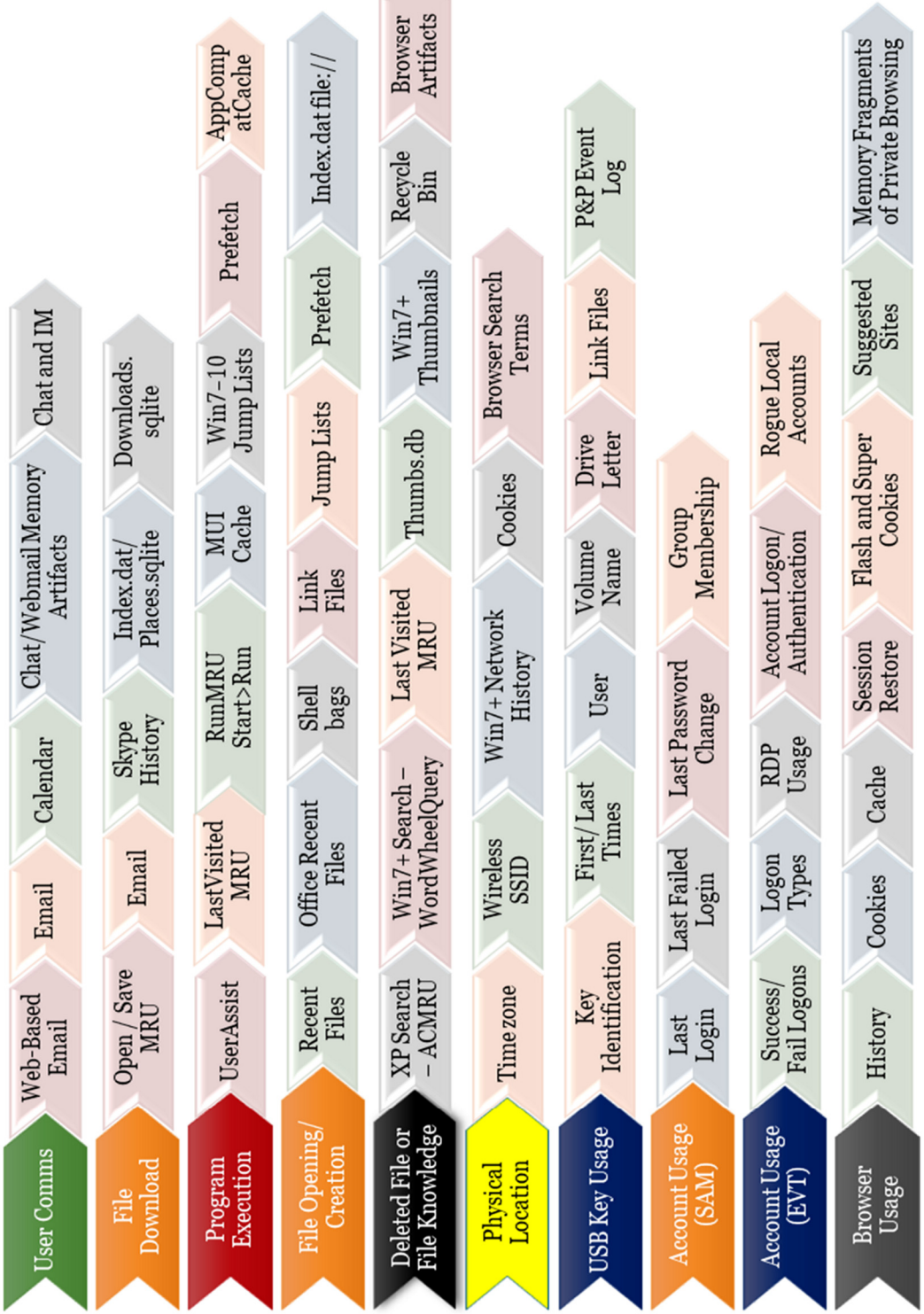
There is a tremendous amount of knowledge required to accomplish and analyze a super timeline effectively. The three core areas of focus are filesystem metadata, windows artifact data, and Windows registry information. Understanding all three areas and how they interrelate is a skill worth working towards.

The stronger your forensic background, the more information you can mine from timeline analysis. If you do not know the significance of a LNK file, then you understandably will not be able to use it within a timeline as an indication of file or folder opening. In this section we will start slow, focusing initially on filesystem metadata and then eventually adding in the rest. If you are new to forensics, it would be worth your time to review SANS Windows Forensic Analysis Poster, much of which is included in the following notes pages. Eventually adding the SANS FOR500 Windows Forensics course to your repertoire will provide a huge boost to your timelining capabilities.



Forensic analysis is very important in this phase of the investigation to identify the patterns of attacker activity. As seen on this slide, we have an enormous set of forensic artifacts used to detect activity of interest on a system. Forensic skills for intrusion scenarios are very similar to those required for tracking other crimes.

This class assumes some familiarity with what we call, "conversational forensics." For example, if we are determining whether malware was executed on a machine, I might mention, "I looked at the prefetch folder for malware execution and also found nothing in the UserAssist key. However, I did find some evidence in the Shellbags that might be useful for our damage assessment." Conversational forensics means you are familiar with most of the terms, and although you might not have them all memorized, a quick mention of the artifact reminds you of what it tells us. If you are not at this level yet, never fear! The Windows Forensic Analysis poster from the SANS FOR500 is here to help. You can find it online, in your class Dropbox, and in the Resources section of your VM-based workbook. It is a terrific resource to reference as we go through this class and when you perform these investigations for real. Some of the techniques taught in this class become even more powerful as you develop a deeper forensic skillset. If you are weak in this area, SANS FOR500 is a great follow-on when you are ready to take your deep-dive forensic skills to the next level.



The Pivot Point

Challenge: Where do I begin to look?

- Use your scope and case knowledge to help form that answer
- A timeline has many places where critical activity has occurred
- Called a timeline pivot point

Pivot Point: Point used to examine the temporal proximity in the timeline

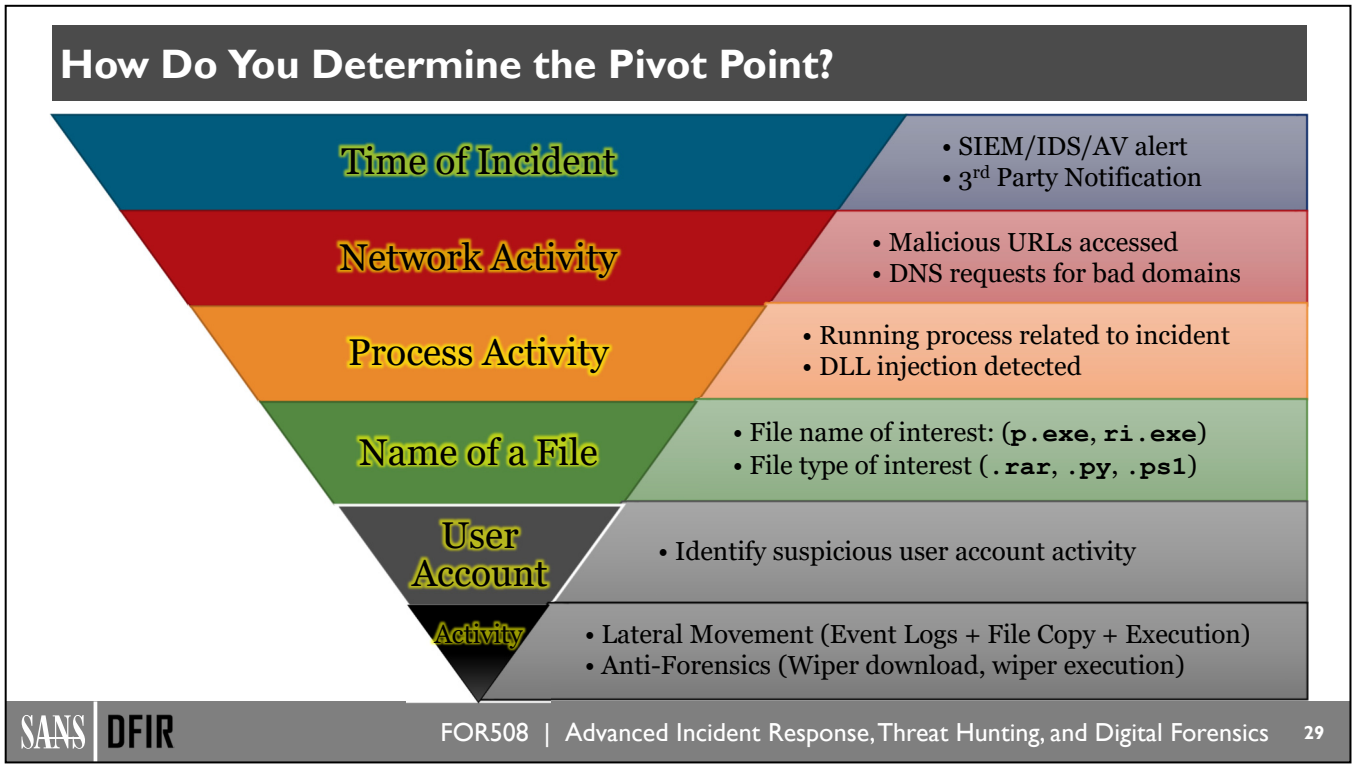
- Temporal Proximity: What occurred immediately before and after a specific event?

Why a pivot point?

- Use the pivot to look before and after in your timeline to get a better idea of what happened on the system
- Example: Program execution followed by multiple writes to C:\Windows\System32 and updating registry entries
- You can also use the "pivot point" to help identify potential malware by finding suspicious files and finding how they interact with the system via the timeline

Proper analysis is the essential component of successful investigations. We don't recover a bunch of artifacts, identify some deleted items, paste them into a report, and hand them to a prosecutor or management. We need to recover the artifacts and analyze the data they hold to provide a clear picture of what users were doing, when they were doing it, why, and many other details that can go unnoticed without a trained eye examining them.

Timeline analysis can be overwhelming at first. There are so many artifacts, entry types, and fields you must master, often resulting in a glassy-eyed analyst staring at the screen trying to make heads or tails of a data set. The secret to success is having a good starting point. To get through the millions of lines of data present in a timeline you must have a good idea of what you are looking for. We call this a "pivot point." This is a relevant point (file, time, artifact) found in the timeline that can be examined to determine what happened before and after that instance. Many refer to this as a "Temporal Proximity" examination.



This chart contains only a small list of possible pivot points available to be used in timeline analysis. An analyst might start with the estimated time of the incident as provided by a security tool and examine what happened on the system at that time. In other instances, such as intellectual property theft or investigations with known malware, it is useful to pivot around a specific filename within the timeline. Related to this is looking for a specific file type (for example, Excel spreadsheets or RAR archives). What about any newly created .exe and .dll files in the C:\Windows\System32 folder? Or event log entries indicating suspicious account activity and lateral movement to a C$ administrative share?

The number of pivot points is endless and driven by whatever knowledge you have of the investigation. As you learn more, you can identify additional pivot points. For each pivot point, plan to review activity both before and after, as it is rare to stumble upon just a single artifact of interest on a system.

Timeline Context Clues

- Recovering a single artifact is similar to recovering a single word
- Seeing the context surrounding the artifact is needed to accurately use timeline
- Example -> ***sweet***
  - **1.*"Sarah is such a sweet little girl; she is always looking after her brother."***
    - Sweet = kind and friendly
  - **2.*"This tea is too sweet for me to drink! How much sugar is in it?"***
    - Sweet = a taste similar to sugar

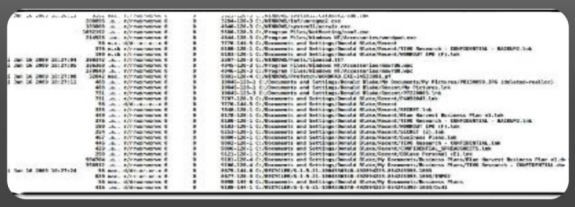
Timeline analysis is unlike traditional data extraction focusing on recovery of a single artifact of data. Identifying a single artifact is simply not enough to understand the part it plays in the larger context of an investigation. This concept is known as artifact temporal proximity. A single artifact might not make sense in the context presented alone, but in the middle of other artifacts it might take on a brand-new meaning.

To highlight what we mean, we can describe the artifact we recover as a single word in a sentence. In this case, we could recover the word "sweet". As we know, the utilization of the word sweet has multiple meanings.

1 - *"Sarah is such a **sweet** little girl; she's always looking after her brother."*
Sweet = kind and friendly.
2 - *"This tea is too **sweet** for me to drink! How much sugar is in it?"*
Sweet = a taste similar to sugar.

The same can occur in timelines with a single artifact. Finding a single artifact is not enough; the analyst needs to examine the temporal proximity of the artifact to determine more detailed information as to what it might represent. You should look at the artifacts in your timeline before and after. You might see evidence of execution of a program, a file being opened, or files deleted. Artifacts do not simply appear or launch themselves as programs without additional interaction. Like Mufasa said in the Lion King, "Everything is connected." The same is true here.

Comparison of Timeline Capabilities



Filesystem: `fls` or `MFTECmd`

- Filesystem metadata only
- More filesystem types
  - Apple (HFS)
  - Solaris (UFS)
  - Linux (EXT)
  - Windows (FAT/NTFS)
  - CD-ROM
- Wider OS/Filesystem capability

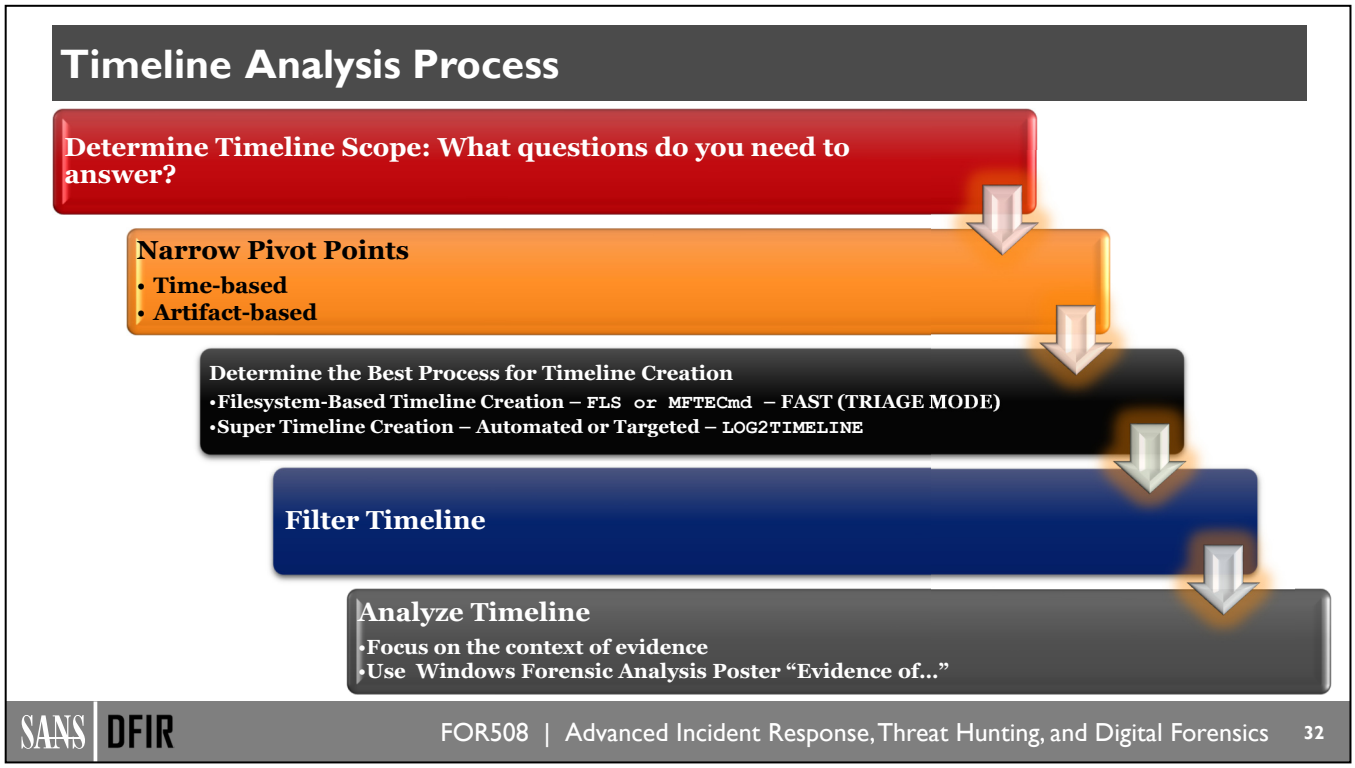
Super Timeline: `log2timeline`

- Obtain everything (Kitchen Sink)
  - Filesystem metadata
  - Artifact timestamps
  - Registry timestamps
- Windows, Linux, and Mac

We will cover two different methods to create a timeline of events on a system, using the tools **`fls`**, **`MFTECmd`**, and **`log2timeline`**.

The original filesystem focused method uses the Sleuthkit tool **`fls`** or newer **`MFTECmd`** to extract only the metadata from the filesystem. Name, path, timestamps, and file size are primary components of a filesystem timeline. These timelines are simple and fast to generate, making them ideal for rapid analysis or analysis at scale. They also provide the greatest filesystem flexibility, capable of extracting timeline data from Apple, Solaris, Linux, CD-ROMS, and of course Windows-based filesystems.

Super timelines extract a much wider set of data from the target system. In addition to filesystem metadata, they often include operating system artifacts, logs, browser activity and specialized artifacts like the Windows registry. The most famous super timeline tool is **`Plaso`**, which is commonly interacted with via the tool **`log2timeline.py`**. While very powerful, super timelines can take a long time to create and the amount of data they contain can be overwhelming. For years Plaso was focused only on Windows forensics, but recent additions to the project have added significant support for Linux, Android, and Mac systems.



Determine Timeline Scope through analysis of the key questions and the case type. Generally, you can identify that the activity occurred between date A and date B. Narrowing the scope will be important in managing your overall data.

Narrow Pivot Points through determining the closest time around when you think the incident occurred (time-based) ,or identifying a key file, user account, or other artifact that might have been used in the activity in question (artifact-based).

Determine the Best Process for Timeline Creation by looking at what data sources you need. A super timeline is preferred if you do not know what you are looking for and you might need to include everything. However, if you can predict the data necessary to solve your case, a targeted super timeline or filesystem timeline might suffice.

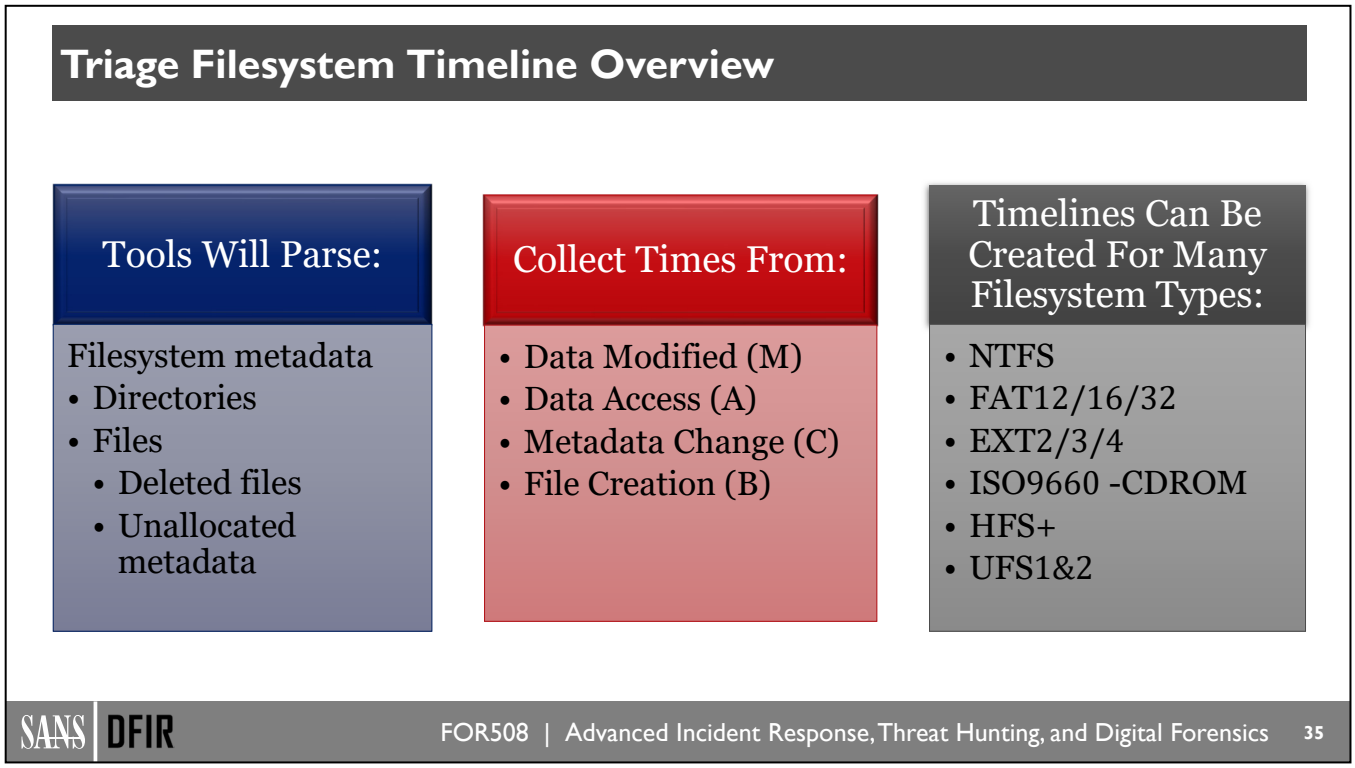
Filter Timeline using your scope, de-duplicate, and eliminate data you do not need to examine. Consider using keywords to identify relevant data (pivot points) for your analysis.

Analyze Timeline through **focusing on the context of evidence** discovered. Look before and after each pivot to determine user activity. **Use the Windows Forensic Analysis Poster "Evidence of" items** to help with analysis.

This page intentionally left blank.

Filesystem Timelining

This page intentionally left blank.



The original and most common type of timeline is the Filesystem Timeline. The filesystem timeline collects data from all the files and directories in a volume. This will include both allocated and unallocated metadata structures, with the latter providing knowledge of deleted and orphan files within the filesystem.

Every filesystem uses different types of timestamps. The most common timestamps record data modification (m-modification), data access (a-access), metadata change time (c-change), and file creation (b-birth). The combination of these timestamps will mean many things and could possibly tell the investigator when a file was created on a volume, copied to a location, or deleted.

A big advantage of filesystem timelines is their ability to be created across a wide range of filesystem types. You never know if you might be examining a GPS device embedded with a Linux filesystem or a dual-boot system that uses both NTFS and HFS+ (Mac in bootcamp mode). The filesystem types that can be parsed with our current filesystem timeline tools include:

- NTFS
- FAT12/16/32
- EXT2/3/4
- ISO9660 -CDROM
- HFS+
- UFS1&2

Windows NTFS Timestamps

- **M** - Data Content Change Time
  - Time the data content of a file was last modified
- **A** - Data Last Access Time
  - Approximate Time when the file data was last accessed
- **C** - Metadata Change Time
  - Time this MFT record was last modified
- **B** - File Creation Time
  - Time file was created in the volume

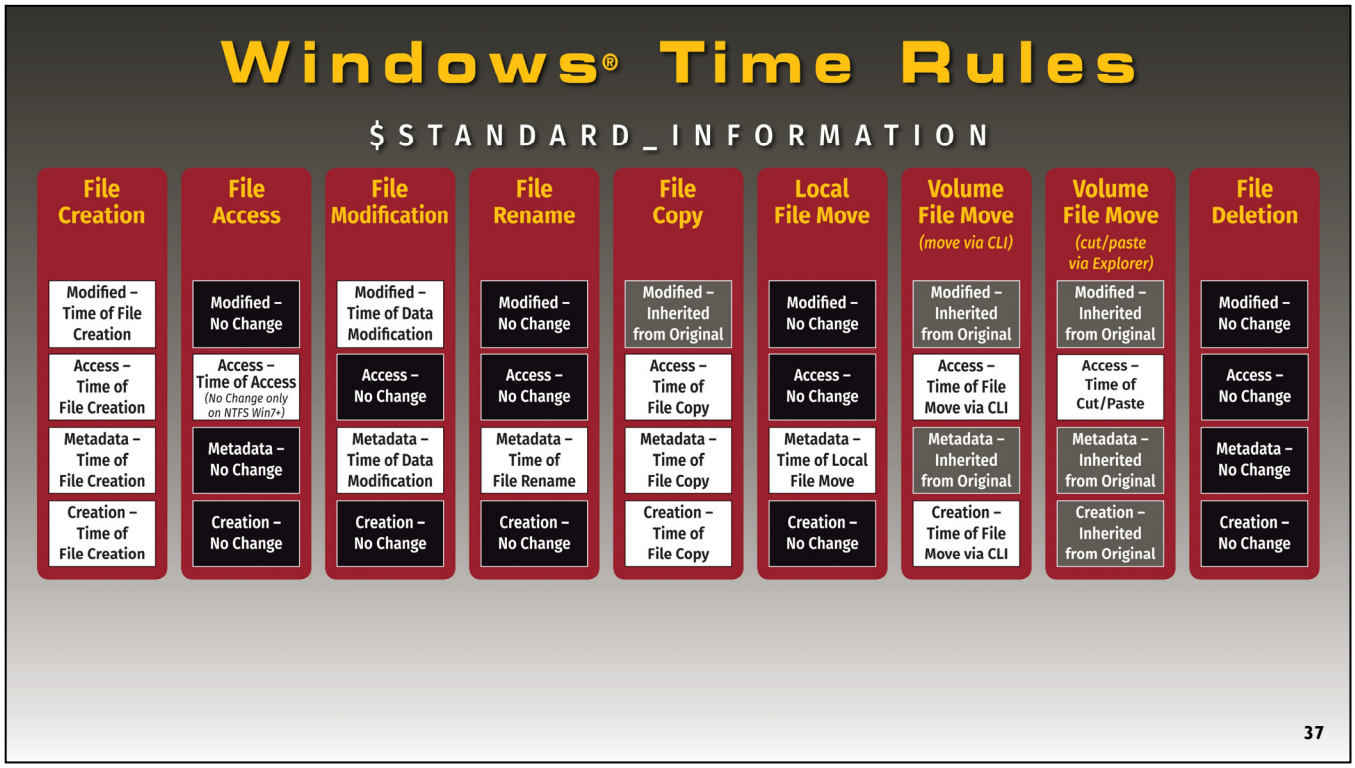
File System	Time Stored	Time Resolution	M	A	C	B
NTFS	UTC	100 ns since Jan 1, 1601	Modified	Accessed	MFT Modified	Created

NTFS stores four significant filesystem times for files and directories: last modification time (M), last access time (A), last modification of the MFT record (C), and file creation time (B). The hardest timestamp for many students to grasp is metadata change time (C). Changes to this timestamp occur when a file is renamed, the file size changes, security permissions update, or if file ownership is changed.[1] Access (A) times have also been historically difficult to interpret because so many actions can lead to an update of this timestamp. Even Windows has historically not given much priority to this timestamp with some versions of the operating system delaying updates to the last access time by up to 1 hour. Furthermore, some Windows versions do not update last access timestamps at all. Unless you have a very specific reason to do otherwise, we recommend primarily focusing on the modified (M) and created (B) times in your investigations. These two timestamps are well understood and well suited to answering most time-based forensic queries.

Luckily, the NTFS filesystem stores time values in UTC format, so they are not affected by changes in time zone or daylight savings time. Amazingly, this is not always the case with other file systems. The FAT filesystem stores time values based on the local time of the computer. So as an example, a file that is saved at 3:00 PM PST in California is seen as 6:00 PM EST in New York on an NTFS volume, but it is (incorrectly) seen as 3:00 PM EST in New York on a FAT volume.

The time format used by NTFS is a 64-bit FILETIME data structure, storing timestamps as the number of hundreds of nanoseconds since 12:00 Jan 1, 1601, UTC.[2] While this seems unusual, the exceptionally high time resolution is a boon to Windows forensics (UNIX stores dates as the number of seconds since Jan 1, 1970). A lot can happen on a system within one second, so the extra granularity can help put findings into proper context.

[1] File Times (Windows) http://for508.com/nep4l
[2] FILETIME structure (Windows) http://for508.com/sucit



This slide has been a work in progress for several years. Tracking the vagaries of different actions on different timestamps across different versions of Windows is enough to drive anyone to madness. That being said, this slide contains a reasonable expectation of how different actions affect timestamps under normal conditions. We recommend using it as a guideline and not gospel truth because there are hundreds of different ways to do the same thing in Windows and not all of them have been tested. If you ever find yourself in a situation where the provenance of a timestamp is crucial, we always recommend testing your hypothesis (multiple times) on a close approximation to the original system.

Notice how each action affects the four timestamps present in Windows NTFS file systems (MACB). Several of the results make sense. When a file is created, its timestamps must be set to something, so all four timestamps are set to the time of creation. File access has been disabled since Windows Vista but appears to be re-enabled in some versions of Windows 10 and 11 (we won't spend much time on this because we recommend ignoring access times since they are so inconsistent). File modifications change the data layer of a file and the file size, so both M and C times are updated. A file rename and a local file move only really affect the metadata of a file (the name and parent folder respectively), so only C time is updated. Windows has no deletion time and hence no timestamps are updated when a file is deleted.

It gets a little more interesting when looking at file copying and volume file moves. A mistake many investigators make is assuming disabled last access times result in zero updates to a file's last access time (the default post-Vista). This is incorrect. The destination file's last access time does update during a file copy or volume move. The most important pattern to recognize on this slide is modified times being inherited from the original file, particularly with a copy and a volume file move via the command line (CLI). In these two situations, we will have a unique pattern – the files will have a modified time pre-dating their creation, which should not be possible. This pattern is an excellent indicator that the files came from somewhere else, and the creation time tells us when the copy or move occurred. Since filesystems do not tend to have "copy" artifacts, this strange set of timestamps is a very important indicator to understanding the movement of files. Interestingly, timestamps update differently for moves conducted via a cut and paste in the GUI desktop versus the command line, a great example of the complexity of timestamp updates.

Windows® Time Rules

$STANDARD_INFORMATION

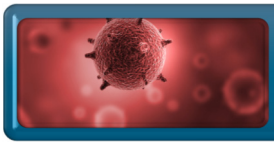
File Creation	File Access	File Modification	File Rename	File Copy	Local File Move	Volume File Move (move via CLI)	Volume File Move (cut/paste via Explorer)	File Deletion
Modified – Time of File Creation	Modified – No Change	Modified – Time of Data Modification	Modified – No Change	Modified – Inherited from Original	Modified – No Change	Modified – Inherited from Original	Modified – Inherited from Original	Modified – No Change
Access – Time of File Creation	Access – Time of Access (No Change only on NTFS Win7+)	Access – No Change	Access – No Change	Access – Time of File Copy	Access – No Change	Access – Time of File Move via CLI	Access – Time of Cut/Paste	Access – No Change
Metadata – Time of File Creation	Metadata – No Change	Metadata – Time of Data Modification	Metadata – Time of File Rename	Metadata – Time of File Copy	Metadata – Time of Local File Move	Metadata – Inherited from Original	Metadata – Inherited from Original	Metadata – No Change
Creation – Time of File Creation	Creation – No Change	Creation – No Change	Creation – No Change	Creation – Time of File Copy	Creation – No Change	Creation – Time of File Move via CLI	Creation – Inherited from Original	Creation – No Change

Additional Time Rule Exceptions

Applications
- Office products
- WinZip

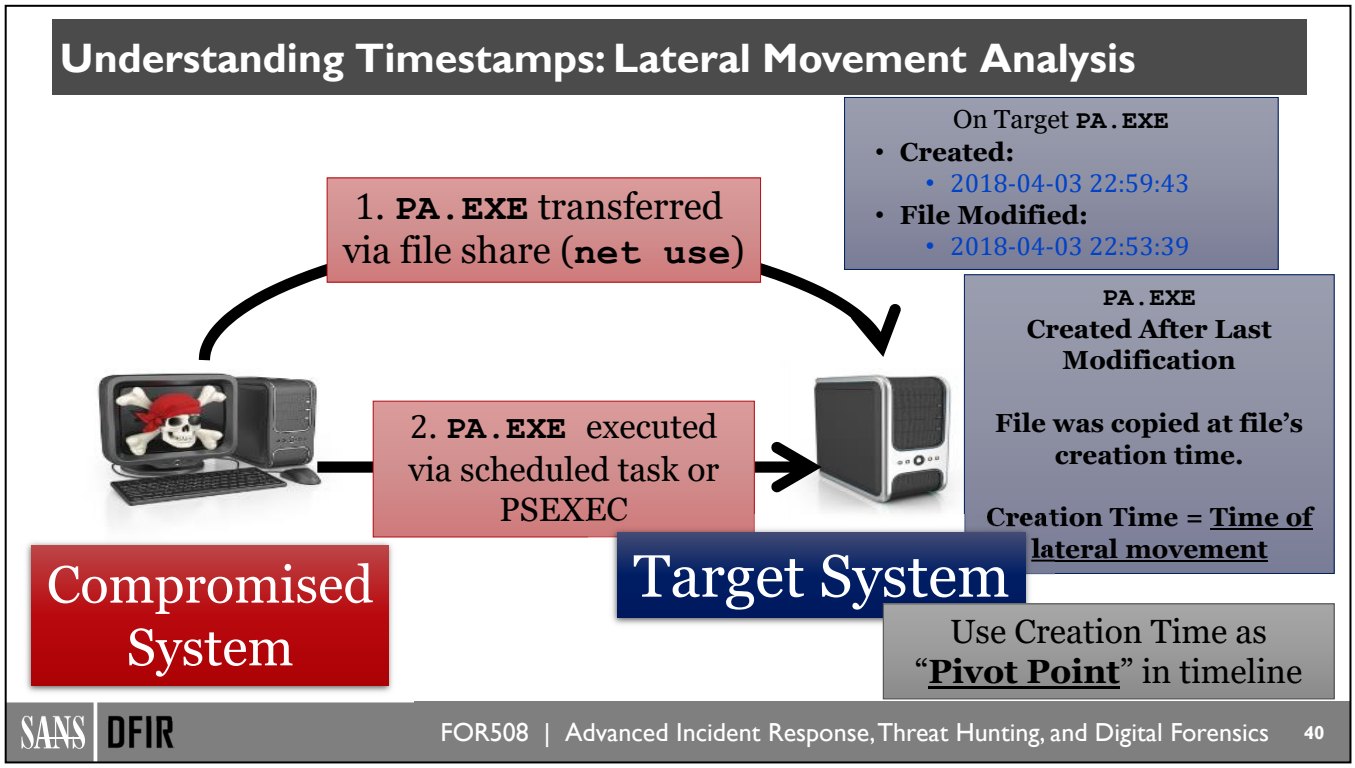
Anti-Forensics
- Timestomp
- Touch
- Privacy cleaners

Archives
- ZIP, RAR, and TGZ
- Retains original date/timestamps
- Usually affects modified time only



Scanning
- Depends on how well the A/V is written

Modern operating systems and the applications running on top of them are extremely complicated. There are myriad actions that might change times in such a way as to break the "standard" time updating rules. Here we try to collect some of the most common offenders. Microsoft Office has long had the habit of updating access times, even when access time updates are turned off in the registry. Malware and anti-forensic tools can leverage file system APIs to change timestamps to better blend in with existing files or destroy evidence. Most archiving software backdates the modification time of unzipped files to the time the dates were previously when the archive was created (making it look like a file copy). Finally, your own security software might be scanning or interacting with files and causing timestamp updates. Many anti-virus products used to update access timestamps every day when they performed their scans. In summary, timestamp interpretation should always be done in context with any other actions that could help explain the changes. When in doubt, test your hypothesis!



One of the more interesting aspects of timestamp analysis is timestamps adhere to the same rules even when communicating over the network via something like a network share (assuming both filesystems are NTFS). A common lateral movement technique is to transfer a file to the targeted host and execute it via a scheduled task, WMI command, or the PsExec tool. Many of these techniques employ a "net use" session, connecting to the targeted filesystem using SMB with valid credentials. The file is "copied" to the remote system and timestamps are transferred along with the file (SMB file transfers seen in network PCAP traffic also show these timestamps). From the point of view of the operating system, a remotely mounted filesystem looks nearly identical to a locally mounted one. This is one of the reasons the "mountpoints2" NTUSER.DAT registry key under each user account is a good source to identify both local and remotely mounted volumes.

When a file is copied over the SMB connection of a file share, the modification time will be inherited from the original, as we would expect. A new creation time will be assigned to the transferred file. Thus, the modification time will pre-date the creation (indicating a copy) and the creation time is the time the file was copied to the target host! This is a tremendous win and very useful for tracking file movements. As an example, how often would you expect an .exe file to exhibit characteristics of copying? This creation time, or "time of file copy", can also be used as a "pivot point" when analyzing the remote system's timeline. Parsing nearby event log entries should show the associated accounts used and application execution events might provide more details into any malware executed. The bottom line: finding clever pivot points can come from a simple understanding of the mechanics of filesystem timestamps.

Understanding the Filesystem Timeline Format

To interpret a timeline you must understand the columns:

- Time: All entries with the same time are grouped
- macb: Indication of timestamp type
- File Size
- Permissions *(Unix only)*
- User and Group *(Unix Only)*
- Meta: Metadata address ($MFT record number for NTFS)
- File Name
  - Deleted files are appended with "(deleted)"

Timestamp	macb	File Size	Meta	File Name
2019-10-03 16:20:37	m.c.	20452	80932	C:\calc.exe
2019-11-04 18:46:51	.a.b	20452	80932	C:\calc.exe

Learning to interpret a timeline takes practice. The "body" file format of a filesystem timeline has up to eight columns (five are seen here). The first column represents the timestamp itself delineated to the closest second. A lot can happen on a system within a second and thus events happening very close to one another (within the second granularity) may be out of order. This is usually unimportant or easy for the analyst to discern.

The "macb" column is by far the hardest (and most important) to understand. This column tells us the specific time value that relates to the timestamp in column one. And it can be more than one. A combination of the letters M,A,C, and B will be present in the column, representing modification, access, metadata change, and birth. In this example on the slide, we see two timeline entries for the same file (C:\calc.exe). The first entry has the value "**`m.c.`**" in its "macb" column. This means the modification and metadata change times for this file are the same, and both are represented by the time in column one (2019-10-03 16:20:37). Similarly, the second line on the slide indicates the access and birth times of the file are the same, via the "macb" column value of "**`.a.b`**". This could be the hardest part of timelines to grasp because there are four timestamps per file, you could have up to four separate entries in your timeline for each file. In our example, we have a total of two entries in the timeline because the modification and metadata change times match, while the access and birth times also match, but at a different time. The entries in the "macb" column will always be in the same order, and any missing timestamps are represented by a period, "`.`" value. Some additional examples:

- **`m.cb`** (modified, metadata changed, birth)
- **`.a..`** (accessed)
- **`mac.`** (modified, accessed, metadata changed)

Remember, we only have the last time for each of these timestamps. As an example, we only have the last modification time (M) for a Word document, not every time it was modified. This means we do not have historical timestamp information, and this is one example of how evidence can age out over time (i.e., the next time that Word document gets updated, its M time will also get updated, erasing the previous modification time).

File size can be helpful when trying to match files with the same size or when looking for very large files like archives or encrypted volumes.

The “Meta” column is the metadata address for the data structure where the data was sourced. In the case of Windows NTFS, this column represents the $MFT record number. In the case of a Linux filesystem this would represent the inode number. We will be using this information later when we dig deeper into MFT records.

The “File Name” column contains full path information for the file. Any currently unallocated file will have (deleted) appended to this column value. This column is one of the most commonly used for filtering, given the file name and path information contained.

Security and ownership information for the file can be found in the “Permissions” and “UID”, and “GID” columns. These columns are only relevant for Unix-based filesystems as Windows does not represent permissions and ownership in this way. We typically hide these columns during the analysis of Windows systems.

Create Triage Timeline Bodyfile Step 1: MFTECmd.exe

```
C:\> MFTECmd.exe -f "E:\C\$MFT" --body "G:\timeline" --bodyf mft.body --blf --bdl C:
```

```
MFTECmd.exe -f "<file>" --body "<dir>" --bodyf file.body --blf --bdl C:
-f "<filename>"          = $MFT|$J|$Boot|$SDS to process
--csv "<dir>"            = Dir to save CSV (tab separated)
--csvf name              = Filename to save CSV
--body "<dir>"           = Dir to save CSV
--bodyf name             = Filename to save CSV
--bdl name               = Drive letter (C, D, etc.) to
                           use with bodyfile
--blf                    = When true, use LF vs CRLF for
                           newlines. Default is FALSE
```


Eric Zimmerman has done it again—producing a program (**MFTECmd)** that is extremely efficient at extracting data from $MFT (Master File Table) files, filesystem journals, and several other NTFS system files.[1] To create our filesystem timeline, we will be using **MFTECmd** to extract data into timeline (bodyfile) format. **MFTECmd** easily extracts the contents of a $MFT file into a format that can be easily filtered and made human-readable using a program we will introduce you to called **mactime**.

MFTECmd can also be used to output MFT metadata to CSV format. The CSV output format allows much more detail to be included and can be a great supplement to your timeline. To see the difference, use "--csv" instead of "--body":

```
MFTECmd.exe -f "E:\C\$MFT" --csv "G:\timeline" --csvf mft.csv
```

MFTECmd.exe --help

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/MFTECmd

Examples: MFTECmd.exe -f "C:\Temp\SomeMFT" --csv "c:\temp\out" --csvf MyOutputFile.csv
MFTECmd.exe -f "C:\Temp\SomeMFT" --csv "c:\temp\out"
MFTECmd.exe -f "C:\Temp\SomeMFT" --json "c:\temp\jsonout"
MFTECmd.exe -f "C:\Temp\SomeMFT" --body "c:\temp\bout" --bdl c
MFTECmd.exe -f "C:\Temp\SomeMFT" --de 5-5

```
Usage:
  MFTECmd [options]

Short options (single letter) are prefixed with a single dash. Long commands are prefixed with two dashes

Options:
  -f <f>          File to process ($MFT | $J | $Boot | $SDS | $I30). Required
  -m <m>          $MFT file to use when -f points to a $J file (Use this to resolve parent path in $J CSV output).
  --json <json>   Directory to save JSON formatted results to. This or --csv required unless --de or --body is
specified
  --jsonf <jsonf>  File name to save JSON formatted results to. When present, overrides default name
  --csv <csv>     Directory to save CSV formatted results to. This or --json required unless --de or --body is
specified
  --csvf <csvf>   File name to save CSV formatted results to. When present, overrides default name
  --body <body>   Directory to save bodyfile formatted results to. --bdl is also required when using this option
  --bodyf <bodyf>  File name to save body formatted results to. When present, overrides default name
  --bdl <bdl>     Drive letter (C, D, etc.) to use with bodyfile. Only the drive letter itself should be provided
  --blf           When true, use LF vs CRLF for newlines [default: False]
  --dd <dd>       Directory to save exported FILE record. --do is also required when using this option
  --do <do>       Offset of the FILE record to dump as decimal or hex. Ex: 5120 or 0x1400 Use --de or --debug
to see offsets
  --de <de>       Dump full details for entry/sequence #. Format is 'Entry' or 'Entry-Seq' as decimal or hex.
Example: 5, 624-5 or 0x270-0x5.
  --dr            When true, dump resident files to dir specified by --csv, in 'Resident' subdirectory. Files will be
named '<EntryNumber>-<SequenceNumber>_<FileName>.bin'
  --fls           When true, displays contents of directory specified by --de. Ignored when --de points to a file
[default: False]
  --ds <ds>       Dump full details for Security Id as decimal or hex. Example: 624 or 0x270
  --dt <dt>       The custom date/time format to use when displaying time stamps. See https://goo.gl/CNVq0k for
options [default: yyyy-MM-dd HH:mm:ss.fffffff]
  --sn            Include DOS file name types [default: False]
  --fl            Generate condensed file listing. Requires --csv [default: False]
  --at            When true, include all timestamps from 0x30 attribute vs only when they differ from 0x10 [default:
False]
  --rs            When true, recover slack space from FILE records when processing MFT files. This option has no
effect for $I30 files [default: False]
  --vss           Process all Volume Shadow Copies that exist on drive specified by -f [default: False]
  --dedupe        Deduplicate -f & VSCs based on SHA-1. First file found wins [default: False]
  --debug         Show debug information during processing [default: False]
  --trace         Show trace information during processing [default: False]
  --version       Show version information
  -?, -h, --help  Show help and usage information
```

[1] https://github.com/EricZimmerman/MFTECmd

Create Triage Timeline Bodyfile Step I (alternate) : `fls -m`

```
fls -r -m C: /cases/cdrive/base-rd01-cdrive.E01 > /cases/cdrive/base-rd01.bodyfile
```

```
fls [options] image [inode]

[Useful Options for fls]
     -d:    Display deleted entries only
     -r:    Recurse on directories
     -p:    Display full path when recursing
     -m:    Display in timeline bodyfile format
     -s <sec>:   Timeskew of system in seconds
```

- The **`fls`** tool allows us to interact with a forensic image as though it were a normal filesystem
- The **`fls`** tool in The Sleuth Kit can be used to collect timeline information from the filename layer
- It takes the inode value of a directory, processes the contents, and displays the filenames in the directory (including deleted items)

The Sleuth Kit (TSK) is one of the original open-source forensic tools and is used by a vast number of other tool suites (free and commercial) to parse file systems. The **`fls`** tool within the TSK suite is designed to extract filename and metadata information for files. By providing the "-m" option, we can tell the tool to output this information in the common "bodyfile" timeline format, which is the first step of timeline creation. There are three different types of data to collect:

1. Allocated files are normal active files one would see when performing a directory listing.
2. Deleted files are unallocated files whose name structures still exist. Depending on the filesystem, the metadata of deleted files can still contain full path information and details such as times and permissions.
3. Orphan files represent data from unallocated metadata structures where the parent folder information no longer exists. We know a file with a specific name and set of timestamps once existed in the filesystem, but do not know in what folder it was present in.

A significant difference between **`fls`** and **`MFTECmd`** is **`fls`** is designed to extract metadata information using an image of a filesystem volume (i.e., the entire C: drive), while **`MFTECmd`** uses just the $MFT file for the C: drive providing timelining capability in times when a disk image is not available or feasible. On the other hand, **`fls`** can parse many more filesystems than just NTFS, while **`MFTECmd`** supports NTFS-only. Note that **`fls`** can also be run against live filesystems, making it potentially useful in various live triage acquisition scenarios.

```
Usage: fls [options] image [inode]

[Useful Options for fls]
        -d:  Display deleted entries only
        -r:  Recurse on directories
        -p:  Display full path when recursing
        -m:  Display in timeline bodyfile format
        -s <sec>:  Timeskew correction for system in seconds
```

Create Triage Timeline Step 2: `mactime`

```
mactime [options] -d -b bodyfile -z timezone > timeline.csv

[Useful Options for mactime]
     -b:        Bodyfile location (data file)
     -y:        Dates are displayed in ISO 8601 format
     -z:        Specify the time zone (see time zone chart)
     -d:        Comma-delimited format

                Optional: Date Range (yyyy-mm-dd..yyyy-mm-dd)
                Example: 2020-01-01..2020-6-01
```

- The mactime tool is a Perl script that takes bodyfile formatted files as input
- It can be given a date range to restrict itself or it can cover the entire time range
- "-z" is the output time zone to use. We highly recommend standardizing on UTC to match other artifacts and eliminate time zone and daylight savings challenges

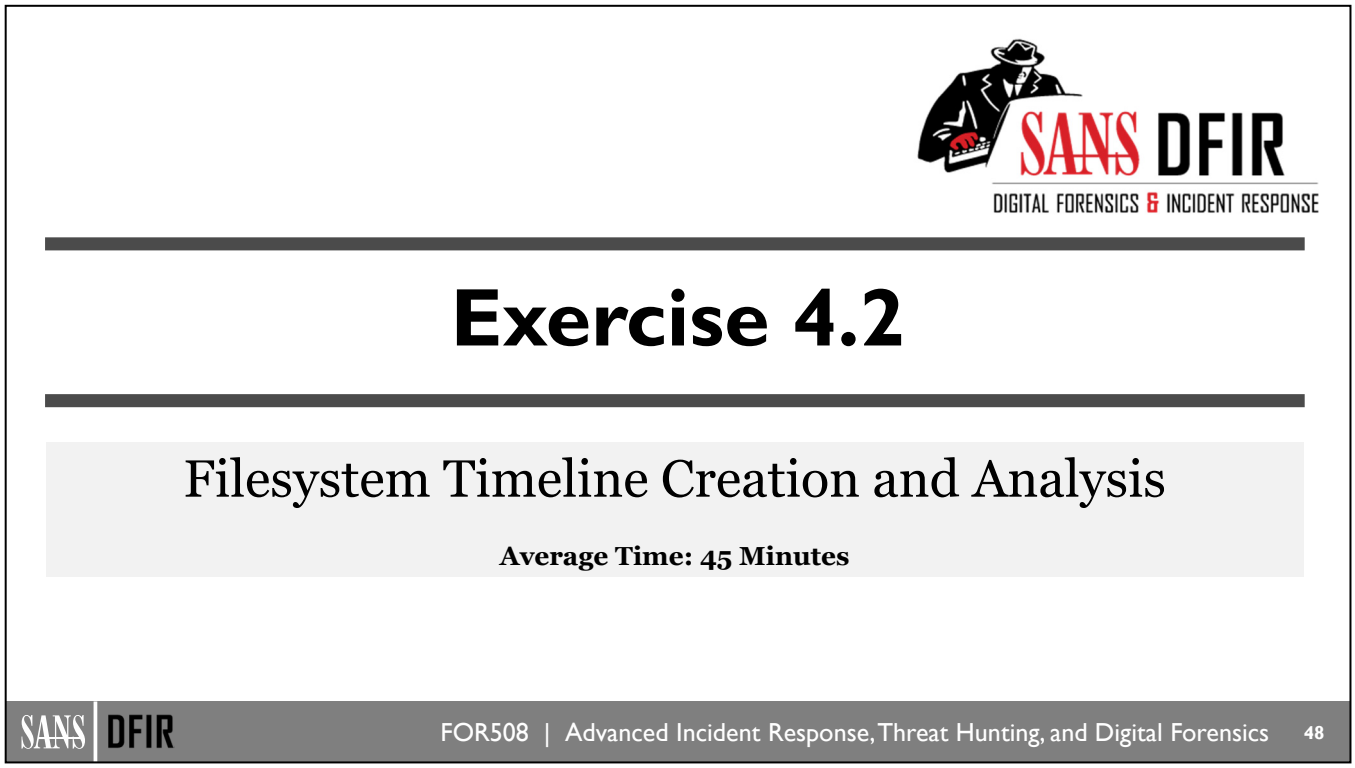
Creating filesystem timelines is a simple two-step process. Once you have a bodyfile containing all the file system metadata (output from either **`fls`** or **`MFTECmd`**), you simply need a tool to make the data human-readable and sort chronologically. **`mactime`** is the tool within The Sleuth Kit (TSK) suite that performs this function.

The **`mactime`** tool takes a bodyfile as input and parses the file to present it into a format that can easily be analyzed by an investigator. You have multiple options available for using **`mactime`**. At a minimum, you need to provide **`mactime`** with the location of the bodyfile with the –b option. The other options allow the investigator some flexibility when working with the resultant data. For example, if you would like to import the data into a spreadsheet or populate a database, you can opt to skip the tabulated (default) format and output in a comma-delimited format (-d). CSV format is what we will use in this class as it can easily be imported into multiple tools and provides a scalability and usefulness for future applications. Timestamps in Windows NTFS are natively stored in UTC format, and we highly recommend standardizing on UTC to match other artifacts and eliminate time zone and daylight savings challenges. However, **`mactime`** is an example of a tool which will attempt to convert your timestamps to your local forensic workstation time by default. To prevent this time conversion, use "-z UTC" to keep timestamps in their native UTC format. Finally, you can provide **`mactime`** a date range to limit your result set. Without the optional date range argument, the mactime tool will output the entire dataset from beginning to end.

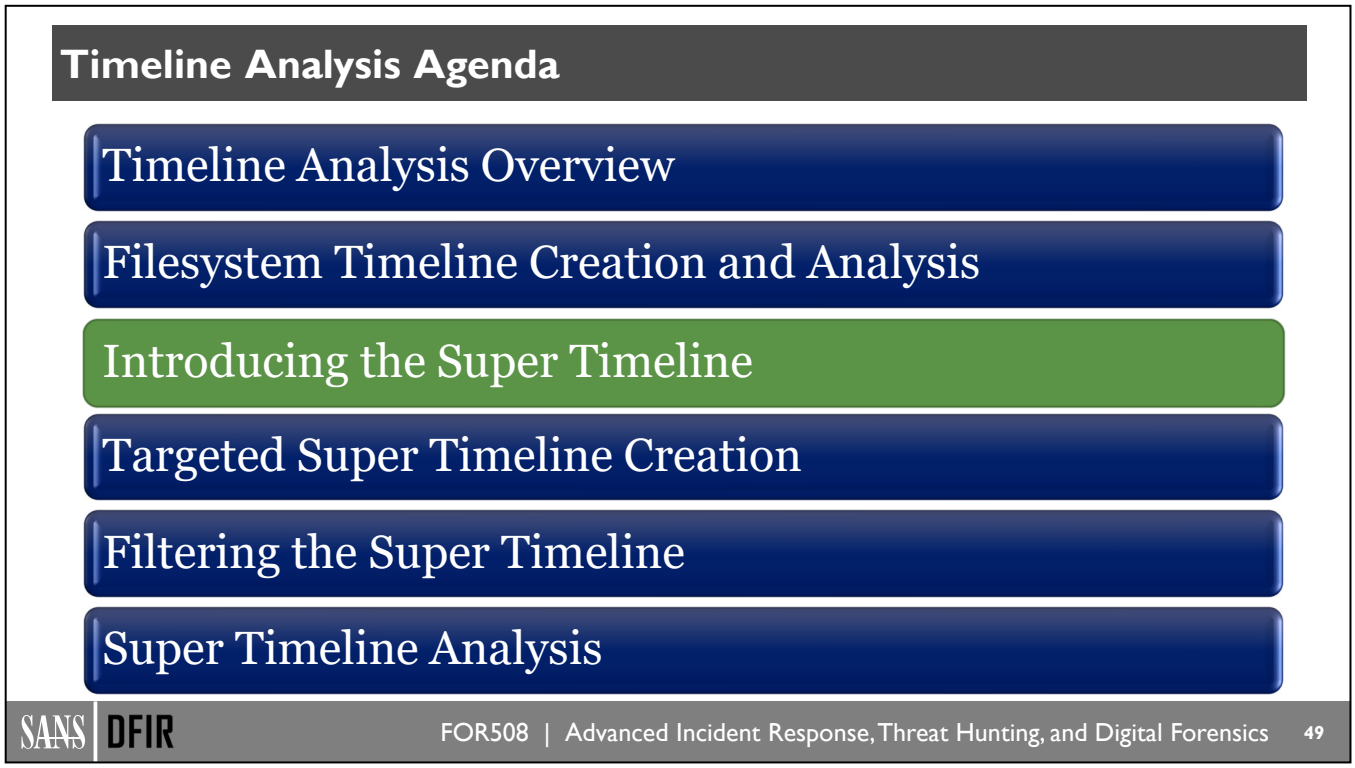
mactime [options] –d -b bodyfile -z timezone > timeline.csv

[Useful Options for mactime]

-b:	Bodyfile location (data file)
-y:	Dates are displayed in ISO 8601 format
-z:	Specify the time zone (see time zone chart)
-d:	Comma-delimited format
	Optional: Date Range (**yyyy-mm-dd..yyyy-mm-dd**)
	Example: **2020-01-01..2020-6-01**



This page intentionally left blank.



This page intentionally left blank.

The Super Timeline w/ `Plaso` and `log2timeline.py`

Information on Plaso co-written by Rob Lee, Kristinn Gudjonsson, and Elizabeth Schweinsberg.

Automated Super Timelining

date	time	MACB	sourcetype	short	filename	inode
1/15/2009	23:59:59	M...	Shortcut LNK	F:/SECRET/SECRET.zip	C:/Documents and Settings/Donald Blake/Recent/SECRET.lnk	7848
1/15/2009	23:59:59	M...	Shortcut LNK	F:/SECRET	C:/Documents and Settings/Donald Blake/Recent/SECRET (2).lnk	8253
1/15/2009	23:59:59	M...	Shortcut LNK	F:/TIVO Research - CONFIDENTIAL - BACKUP2.doc	C:/Documents and Settings/Donald Blake/Recent/TIVO Research - CONFIDE	8180
1/16/2009	13:05:55	M.CB	NTFS $MFT	C:/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1	C:/WINDOWS/pchealth/helpctr/DataColl/CollectedData_177.xml	8175
1/16/2009	17:31:35	..CB	Shortcut LNK	E:/	C:/Documents and Settings/Donald Blake/Recent/DBlake Personal (E).lnk	9121
1/16/2009	17:40:27	M...	NTFS $MFT	C:/Documents and Settings/Donald Blake/My Documents/	C:/Documents and Settings/Donald Blake/My Documents/Business Plans/B	9101
1/16/2009	17:40:27	.A..	Shortcut LNK	E:/Blue Harvest Business Plan v1.doc	C:/Documents and Settings/Donald Blake/Recent/Blue Harvest Business Pl	8178
1/16/2009	17:42:43	.A..	Shortcut LNK	E:/TIVO Research - CONFIDENTIAL.doc	C:/Documents and Settings/Donald Blake/Recent/TIVO Research - CONFIDE	9092
1/16/2009	17:42:44	.A..	Shortcut LNK	F:/TIVO Research - CONFIDENTIAL - BACKUP2.doc	C:/Documents and Settings/Donald Blake/Recent/TIVO Research - CONFIDE	8180
1/16/2009	17:49:52	.A..	Shortcut LNK	E:/CONFIDENTIAL_SPREADSHEETS.zip	C:/Documents and Settings/Donald Blake/Recent/CONFIDENTIAL_SPREADS	9096
1/16/2009	17:50:33	M...	NTFS $MFT	C:/RECYCLER/S-1-5-21-1004336348-492894223-854245398-1	C:/RECYCLER/S-1-5-21-1004336348-492894223-854245398-1003/Dc41/SECRET	9110
1/16/2009	17:50:34	.A..	Shortcut LNK	F:/SECRET/SECRET.zip	C:/Documents and Settings/Donald Blake/Recent/SECRET.lnk	7848
1/16/2009	17:58:20	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Application Dat	C:/Documents and Settings/Donald Blake/Application Data/Skype/dblake_	8177
1/16/2009	17:58:20	MAC.	NTFS $MFT	C:/Documents and Settings/Donald Blake/Application Dat	C:/Documents and Settings/Donald Blake/Application Data/Skype/dblake_	4065
1/16/2009	17:58:20	M.CB	NTFS $MFT	C:/Documents and Settings/Donald Blake/Application Dat	C:/Documents and Settings/Donald Blake/Application Data/Skype/dblake_	8173
1/16/2009	18:10:53	.A..	NTFS $MFT	C:/WINDOWS/system32/msi.dll	C:/WINDOWS/system32/msi.dll	2232
1/16/2009	18:10:53	.A..	NTFS $MFT	C:/Program Files/Google	C:/Program Files/Google	8775
1/16/2009	18:10:54	.A..	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@dc[1].txt	11612
1/16/2009	18:10:57	.ACB	Internet Explorer	visited :Host: us.mg4.mail.yahoo.com	C:/Documents and Settings/Donald Blake/Local Settings/History/History.IE	11576
1/16/2009	18:10:57	M...	Internet Explorer	visited http://us.mg4.mail.yahoo.com/dc/launch?.rand=d	C:/Documents and Settings/Donald Blake/Local Settings/History/History.IE	7849
1/16/2009	18:10:57	.ACB	Internet Explorer	visited http://us.mg4.mail.yahoo.com/dc/launch?.rand=d	C:/Documents and Settings/Donald Blake/Local Settings/History/History.IE	11576 7849
1/16/2009	18:10:57	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@dynamic	9009
1/16/2009	18:10:57	M.CB	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@media.a	9003
1/16/2009	18:10:57	.A..	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@ad.yield	11640
1/16/2009	18:10:57	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@adrevolv	9006
1/16/2009	18:10:57	.ACB	Internet Explorer	visited media.adrevolver.com/	C:/Documents and Settings/Donald Blake/Cookies/index.dat	7855
1/16/2009	18:11:06	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@yahoo[2]	9074
1/16/2009	18:11:07	.A..	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@action.m	13013
1/16/2009	18:11:07	M...	Internet Explorer	visited action.mathtag.com/	C:/Documents and Settings/Donald Blake/Cookies/index.dat	7855
1/16/2009	18:12:02	MACB	Event Log	Application Popup/26;Info;Windows - Virtual Memory Mir	C:/WINDOWS/system32/config/SysEvent.Evt	3524

This is an example of a (partial) super timeline. Notice there are entries from the filesystem (NTFS $MFT), event logs, LNK files, and IE history. Review the timeline. Can you describe what occurred by looking at this timeline?

1. At the top of the output, we see LNK files for a SECRET folder and zip archive along with access to a TIVO research document. These LNK files were modified, which means this is the most recent time these files were opened by the user.

2. What two drive letters are present that could indicate USB/external device activity?

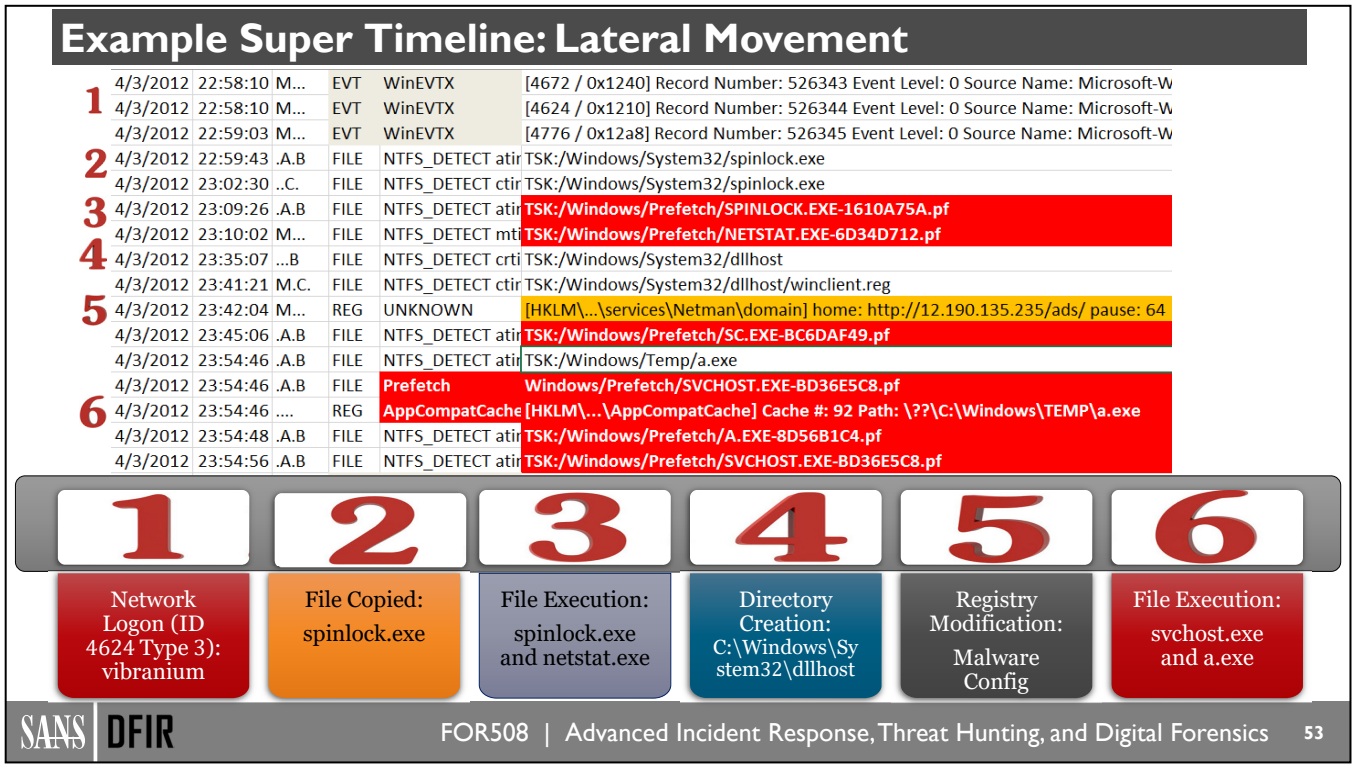
3. Note the creation of a LNK file for "E:\". This could indicate a device was inserted (and opened in something like File Explorer). Several LNK files were created immediately following this event, indicating they were opened from the external device.

4. Can you identify when Internet Explorer was used to surf the web? Notice the clue that this user has a webmail account?

5. Note the event log entry near the bottom. Event logs can be very helpful in gaining further information about system activity.

If you had access to the full timeline, you could tell much more.

date	time	MACB	sourcetype	short	filename	inode
1/15/2009	23:59:59	M...	Shortcut LNK	F:/SECRET/SECRET.zip	C:/Documents and Settings/Donald Blake/Recent/SECRET.lnk	7848
1/15/2009	23:59:59	M...	Shortcut LNK	F:/SECRET	C:/Documents and Settings/Donald Blake/Recent/SECRET (2).lnk	8253
1/15/2009	23:59:59	M...	Shortcut LNK	F:/TIVO Research - CONFIDENTIAL - BACKUP2.doc	C:/Documents and Settings/Donald Blake/Recent/TIVO Research - CONFIDE	8180
1/16/2009	13:05:55	M.CB	NTFS $MFT	C:/WINDOWS/pchealth/helpctr/DataColl/CollectedData_1	C:/WINDOWS/pchealth/helpctr/DataColl/CollectedData_177.xml	8175
1/16/2009	17:31:35	..CB	Shortcut LNK	E:/	C:/Documents and Settings/Donald Blake/Recent/DBlake Personal (E).lnk	9121
1/16/2009	17:40:27	M...	NTFS $MFT	C:/Documents and Settings/Donald Blake/My Documents/	C:/Documents and Settings/Donald Blake/My Documents/Business Plans/B	9101
1/16/2009	17:40:27	.A..	Shortcut LNK	E:/Blue Harvest Business Plan v1.doc	C:/Documents and Settings/Donald Blake/Recent/Blue Harvest Business Pl	8178
1/16/2009	17:42:43	.A..	Shortcut LNK	E:/TIVO Research - CONFIDENTIAL.doc	C:/Documents and Settings/Donald Blake/Recent/TIVO Research - CONFIDE	9092
1/16/2009	17:42:44	.A..	Shortcut LNK	F:/TIVO Research - CONFIDENTIAL - BACKUP2.doc	C:/Documents and Settings/Donald Blake/Recent/TIVO Research - CONFIDE	8180
1/16/2009	17:49:52	.A..	Shortcut LNK	E:/CONFIDENTIAL_SPREADSHEETS.zip	C:/Documents and Settings/Donald Blake/Recent/CONFIDENTIAL_SPREADS	9096
1/16/2009	17:50:33	M...	NTFS $MFT	C:/RECYCLER/S-1-5-21-1004336348-492894223-854245398-1	C:/RECYCLER/S-1-5-21-1004336348-492894223-854245398-1003/Dc41/SECRET	9110
1/16/2009	17:50:34	.A..	Shortcut LNK	F:/SECRET/SECRET.zip	C:/Documents and Settings/Donald Blake/Recent/SECRET.lnk	7848
1/16/2009	17:58:20	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Application Dat	C:/Documents and Settings/Donald Blake/Application Data/Skype/dblake_	8177
1/16/2009	17:58:20	MAC.	NTFS $MFT	C:/Documents and Settings/Donald Blake/Application Dat	C:/Documents and Settings/Donald Blake/Application Data/Skype/dblake_	4065
1/16/2009	17:58:20	M.CB	NTFS $MFT	C:/Documents and Settings/Donald Blake/Application Dat	C:/Documents and Settings/Donald Blake/Application Data/Skype/dblake_	8173
1/16/2009	18:10:53	.A..	NTFS $MFT	C:/WINDOWS/system32/msi.dll	C:/WINDOWS/system32/msi.dll	2232
1/16/2009	18:10:53	.A..	NTFS $MFT	C:/Program Files/Google	C:/Program Files/Google	8775
1/16/2009	18:10:54	.A..	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@dc[1].txt	11612
1/16/2009	18:10:57	.ACB	Internet Explorer	visited :Host: us.mg4.mail.yahoo.com	C:/Documents and Settings/Donald Blake/Local Settings/History/History.IE	11576
1/16/2009	18:10:57	M...	Internet Explorer	visited http://us.mg4.mail.yahoo.com/dc/launch?.rand=dl	C:/Documents and Settings/Donald Blake/Local Settings/History/History.IE	7849
1/16/2009	18:10:57	.ACB	Internet Explorer	visited http://us.mg4.mail.yahoo.com/dc/launch?.rand=dl	C:/Documents and Settings/Donald Blake/Local Settings/History/History.IE	11576 7849
1/16/2009	18:10:57	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@dynamic	9009
1/16/2009	18:10:57	M.CB	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@media.at	9003
1/16/2009	18:10:57	.A..	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@ad.yieldı	11640
1/16/2009	18:10:57	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@adrevolv	9006
1/16/2009	18:10:57	.ACB	Internet Explorer	visited media.adrevolver.com/	C:/Documents and Settings/Donald Blake/Cookies/index.dat	7855
1/16/2009	18:11:06	...B	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@yahoo[2]	9074
1/16/2009	18:11:07	.A..	NTFS $MFT	C:/Documents and Settings/Donald Blake/Cookies/donald	C:/Documents and Settings/Donald Blake/Cookies/donald blake@action.m	13013
1/16/2009	18:11:07	M...	Internet Explorer	visited action.mathtag.com/	C:/Documents and Settings/Donald Blake/Cookies/index.dat	7855
1/16/2009	18:12:02	MACB	Event Log	Application Popup/26;Info;Windows - Virtual Memory Mir	C:/WINDOWS/system32/config/SysEvent.Evt	3524



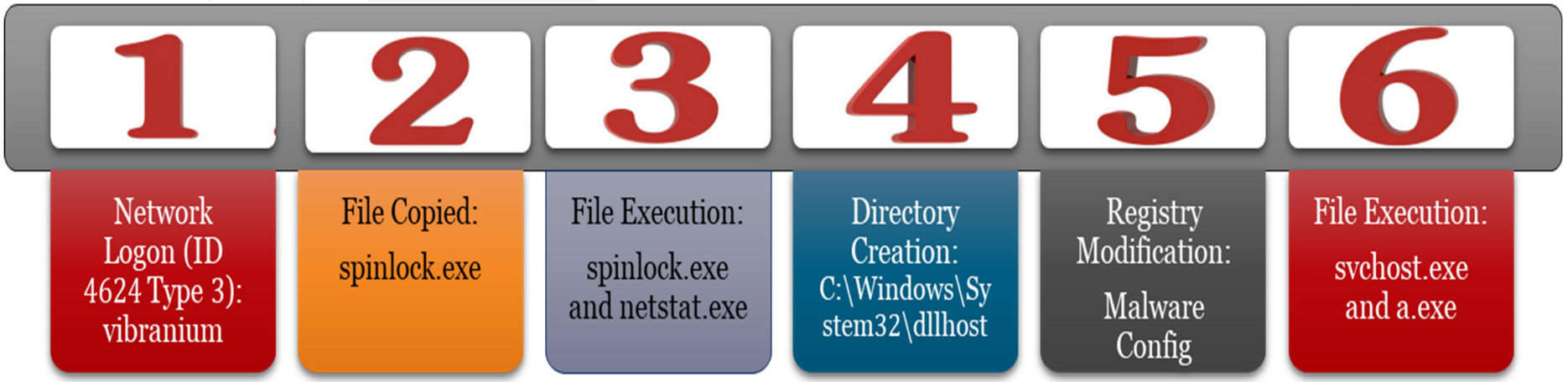
Timeline forensics can result in output like the above with a little work on behalf of the investigator. Notice program execution artifacts are colored in red. URLs are colored in orange. More on that in a moment. The amount of information can be overwhelming, but if utilized properly, can show what happened on a system second by second during an incident. Our forensic knowledge of the filesystem, artifacts, and the registry combined with a timeline can paint a clear and articulate picture of a subject's activities — even if anti-forensics tools are employed.

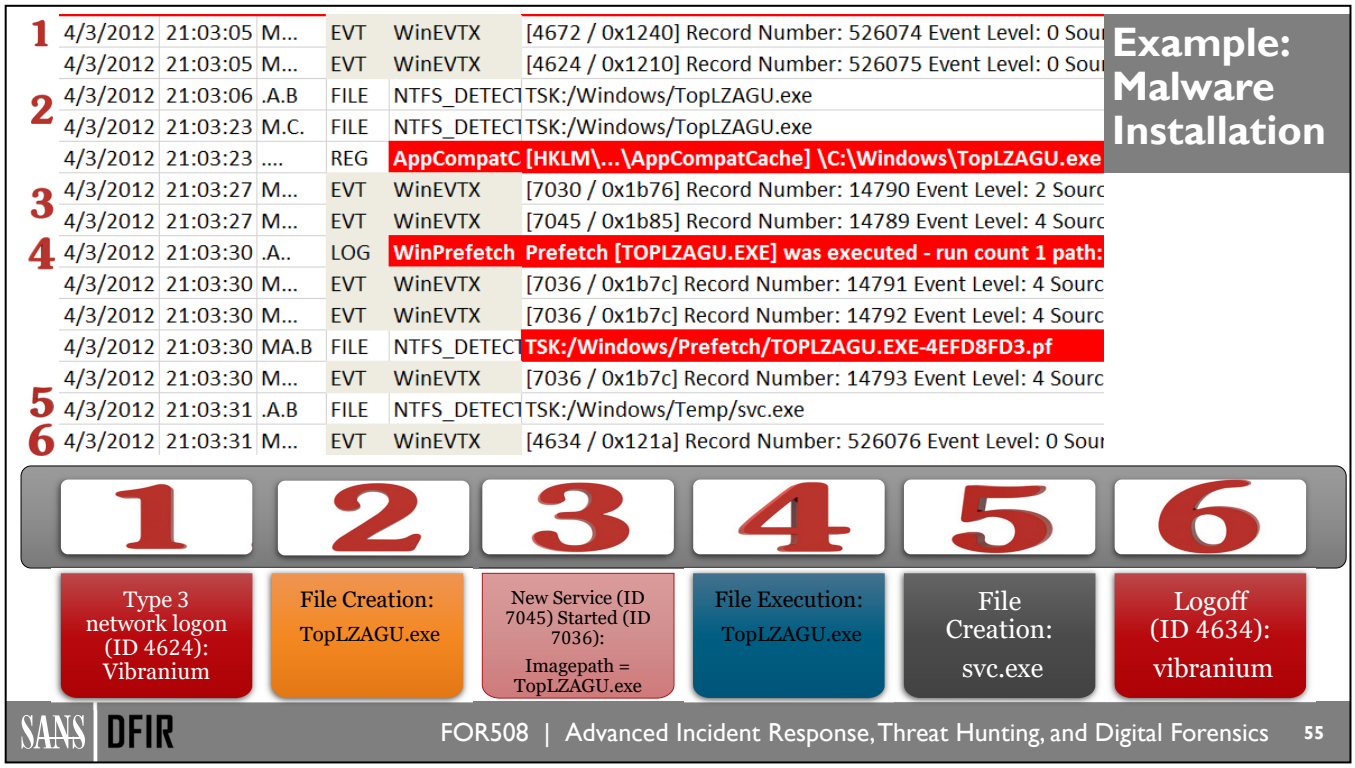
In this example, the creation of a new executable, spinlock.exe in the C:\Windows\System32 folder might draw our interest. Looking above that activity we see evidence of account logon. Below it, evidence of application execution and registry modification. This looks like an interesting pivot point! In a little more detail:

1. The Vibranium account logs in to system as a local administrator (EIDs 4624/4672/4776) at 22:58:10
2. Spinlock.exe is created on local system (Creation Time of Spinlock = 22:59:43)
3. Spinlock and Netstat are executed at ~23:09:26 (-10 seconds) and ~23:10:02 (-10 seconds), respectively
4. C:\Windows\System32\dllhost directory is created at 23:35:07
5. Modification of the \services\Netman\domain registry key (which contains an interesting URL) occurs at 23:42:04
6. Svchost.exe and a.exe are executed for the first time. The latter is highly suspicious, and due to their temporal relationship in the timeline we need to also determine the full path of the svchost file

That is a lot of information in just a small section of a timeline!

	4/3/2012	22:58:10	M...	EVT	WinEVTX	[4672 / 0x1240] Record Number: 526343 Event Level: 0 Source Name: Microsoft-W
1	4/3/2012	22:58:10	M...	EVT	WinEVTX	[4624 / 0x1210] Record Number: 526344 Event Level: 0 Source Name: Microsoft-W
	4/3/2012	22:59:03	M...	EVT	WinEVTX	[4776 / 0x12a8] Record Number: 526345 Event Level: 0 Source Name: Microsoft-W
2	4/3/2012	22:59:43	.A.B	FILE	NTFS_DETECT atir	TSK:/Windows/System32/spinlock.exe
	4/3/2012	23:02:30	..C.	FILE	NTFS_DETECT ctir	TSK:/Windows/System32/spinlock.exe
3	4/3/2012	23:09:26	.A.B	FILE	NTFS_DETECT atir	TSK:/Windows/Prefetch/SPINLOCK.EXE-1610A75A.pf
	4/3/2012	23:10:02	M...	FILE	NTFS_DETECT mti	TSK:/Windows/Prefetch/NETSTAT.EXE-6D34D712.pf
4	4/3/2012	23:35:07	...B	FILE	NTFS_DETECT crti	TSK:/Windows/System32/dllhost
	4/3/2012	23:41:21	M.C.	FILE	NTFS_DETECT ctir	TSK:/Windows/System32/dllhost/winclient.reg
5	4/3/2012	23:42:04	M...	REG	UNKNOWN	[HKLM\...\services\Netman\domain] home: http://12.190.135.235/ads/ pause: 64
	4/3/2012	23:45:06	.A.B	FILE	NTFS_DETECT atir	TSK:/Windows/Prefetch/SC.EXE-BC6DAF49.pf
	4/3/2012	23:54:46	.A.B	FILE	NTFS_DETECT atir	TSK:/Windows/Temp/a.exe
	4/3/2012	23:54:46	.A.B	FILE	Prefetch	Windows/Prefetch/SVCHOST.EXE-BD36E5C8.pf
6	4/3/2012	23:54:46	....	REG	AppCompatCache	[HKLM\...\AppCompatCache] Cache #: 92 Path: \??\C:\Windows\TEMP\a.exe
	4/3/2012	23:54:48	.A.B	FILE	NTFS_DETECT atir	TSK:/Windows/Prefetch/A.EXE-8D56B1C4.pf
	4/3/2012	23:54:56	.A.B	FILE	NTFS_DETECT atir	TSK:/Windows/Prefetch/SVCHOST.EXE-BD36E5C8.pf





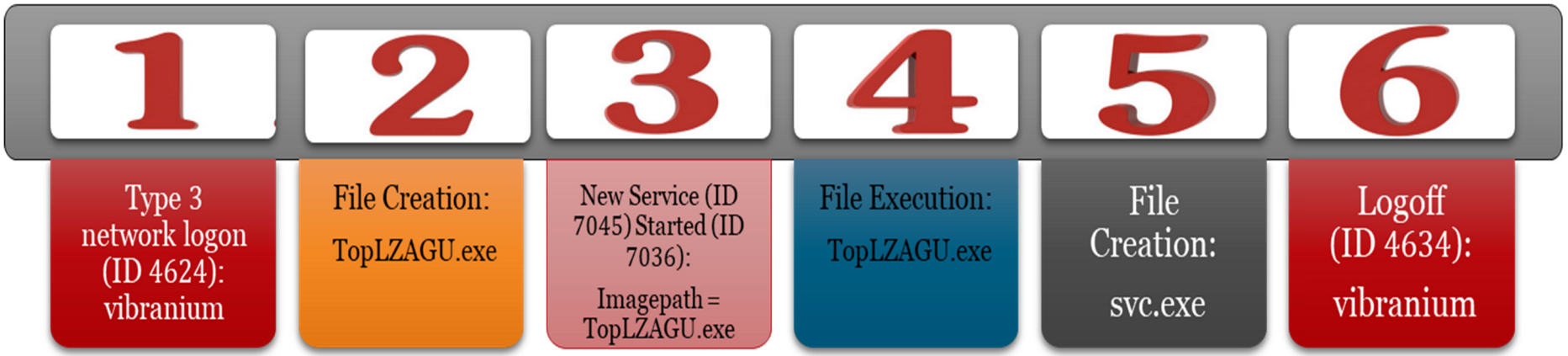
Here is one more example to show the power of super timeline analysis. Here the execution of TOPLAZAGU.EXE caught our attention. That seems like an unusual executable name, and by looking at the entries around it, we can confirm (or refute) our suspicions.

1. Type 3 network logon for an admin account named Vibranium (EIDs 4624/4672). The name and the logon type are included in the description field, but not seen on this slide.
2. File creation: TopLZAGU.exe
3. New service creation (EID 7045): Imagepath = TopLZAGU.exe
4. File execution indicated by Prefetch: TopLZAGU.exe
5. File creation: svc.exe
6. Logoff (EID 4634): Vibranium account

Notice the timing of all these events. That is a significant amount of activity to occur in less than thirty seconds! We might hypothesize these events were scripted or automated as it is unlikely a user could accomplish all of this within that timeframe.

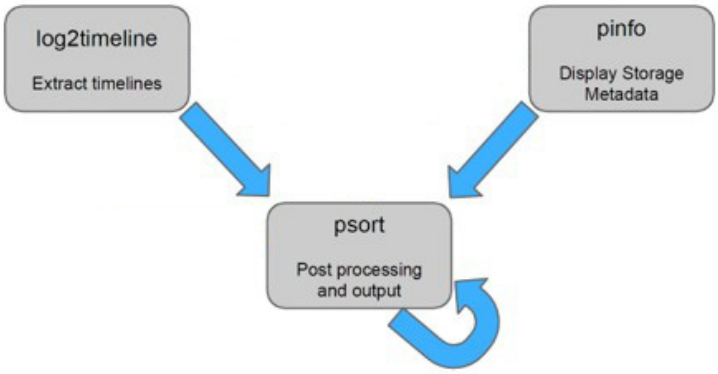
Example: Malware Installation

1	4/3/2012	21:03:05	M...	EVT	WinEVTX	[4672 / 0x1240] Record Number: 526074 Event Level: 0 Sou
	4/3/2012	21:03:05	M...	EVT	WinEVTX	[4624 / 0x1210] Record Number: 526075 Event Level: 0 Sou
2	4/3/2012	21:03:06	.A.B	FILE	NTFS_DETECT	TSK:/Windows/TopLZAGU.exe
	4/3/2012	21:03:23	M.C.	FILE	NTFS_DETECT	TSK:/Windows/TopLZAGU.exe
	4/3/2012	21:03:23	....	REG	AppCompatC	[HKLM\...\AppCompatCache] \C:\Windows\TopLZAGU.exe
3	4/3/2012	21:03:27	M...	EVT	WinEVTX	[7030 / 0x1b76] Record Number: 14790 Event Level: 2 Sourc
	4/3/2012	21:03:27	M...	EVT	WinEVTX	[7045 / 0x1b85] Record Number: 14789 Event Level: 4 Sourc
4	4/3/2012	21:03:30	.A..	LOG	WinPrefetch	Prefetch [TOPLZAGU.EXE] was executed - run count 1 path:
	4/3/2012	21:03:30	M...	EVT	WinEVTX	[7036 / 0x1b7c] Record Number: 14791 Event Level: 4 Sourc
	4/3/2012	21:03:30	M...	EVT	WinEVTX	[7036 / 0x1b7c] Record Number: 14792 Event Level: 4 Sourc
	4/3/2012	21:03:30	MA.B	FILE	NTFS_DETECT	TSK:/Windows/Prefetch/TOPLZAGU.EXE-4EFD8FD3.pf
	4/3/2012	21:03:30	M...	EVT	WinEVTX	[7036 / 0x1b7c] Record Number: 14793 Event Level: 4 Sourc
5	4/3/2012	21:03:31	.A.B	FILE	NTFS_DETECT	TSK:/Windows/Temp/svc.exe
6	4/3/2012	21:03:31	M...	EVT	WinEVTX	[4634 / 0x121a] Record Number: 526076 Event Level: 0 Sou



Introducing Plaso and `log2timeline.py`

Collect Time-Based Events from Mac, Linux, and Windows



Log2timeline was created by Kristinn Guðjónsson as a part of a discussion for his GCFA Gold Certification here at SANS. Prompted by SANS Faculty Fellow Rob Lee for a need to compile a variety of time information into a single file, Kristinn felt a single tool with a variety of options was probably the best bet. What he created was far better and more important than anyone in the forensic community might have imagined. Kristinn's project is incredibly important to the forensic community, as it is the only tool to help parse many of these artifacts, so an investigator has a single point of reference for their case. Kristinn is continually updating this project, so if you have any comments or suggestions, please contribute!

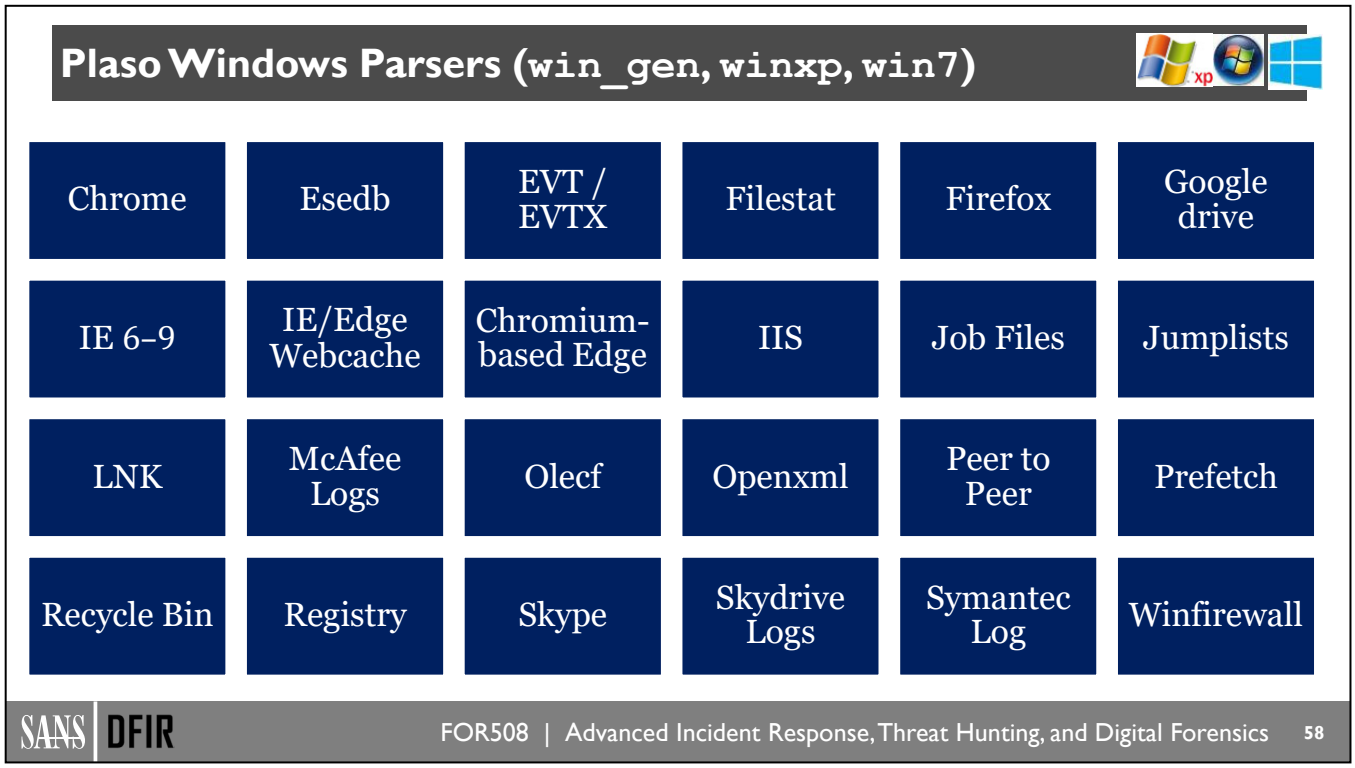
Plaso (plaso langar að safna öllu) is the Python-based backend engine now used for creation of super timelines. We will use the terms Plaso and log2timeline interchangeably throughout this section.

Important Components of Plaso

log2timeline: This is the main single-machine frontend to the Plaso backend. This is the tool that can be used to extract events from a group of files, mount point, or a forensic image and save the results in a Plaso storage file for future processing and analysis.

pinfo: The plaso storage file contains a variety of information about how and when the collection took place. It may also contain information from any preprocessing stages that were employed. pinfo is a simple tool designed to print out this information from a storage database file.

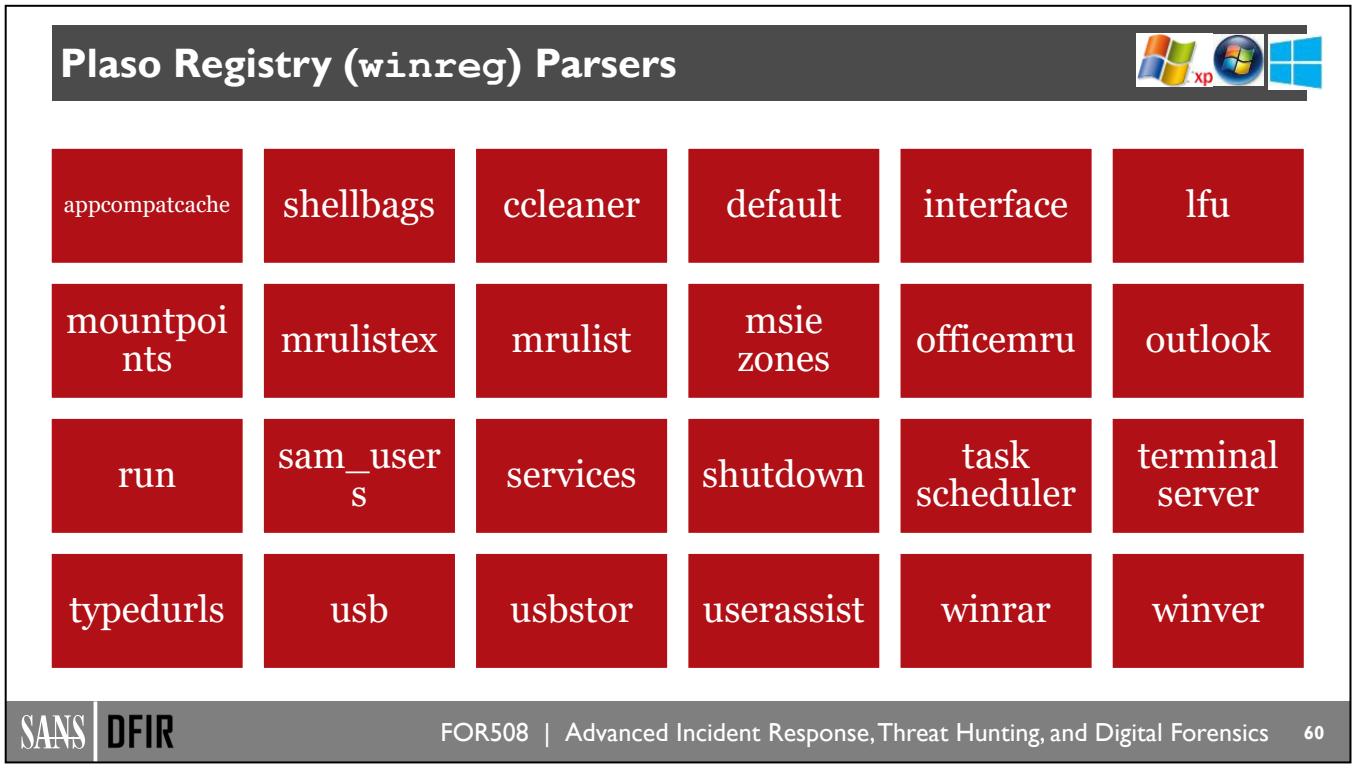
psort: The post-processing tool used to filter, sort, and process the plaso storage file. This tool is used for all post-processing filtering, sorting, and tagging of the storage file. Because the Plaso storage format is not in human-readable format, it is typically necessary to run this tool on the storage file to create useful output.



Plaso/Log2timeline was initially built to scan and extract logs and time-related artifacts from Windows operating systems. It extracts a wealth of information the registry, browser history, shell items, prefetch, and more. These artifacts often make up nearly 99% of the forensic data an investigator needs during an investigation. The power of Log2timeline is in its ability to normalize all of these disparate data sources into one consistent view for the investigator. Bringing many artifacts together in the same view facilitates interpretation and leads to rapid understanding of events that occurred on the system.

Windows Parsers
bencode: Parser for bencoded files.
bencode_transmission: Parser for Transmission bencoded files.
bencode_utorrent: Parser for uTorrent bencoded files.
custom_destinations: Parser for *.customDestinations-ms files.
esedb: Parser for Extensible Storage Engine (ESE) database files.
filestat: Parser for filesystem stat information.
hachoir: Parser that wraps Hachoir.
lnk: Parser for Windows Shortcut (LNK) files.
mcafee_protection: Parser for McAfee AV Access Protection log files.
olecf_document_summary: Parser for a DocumentSummaryInformation OLECF stream.
olecf_summary: Parser for a SummaryInformation OLECF stream.
openxml: Parser for OpenXML (OXML) files.
prefetch: Parser for Windows Prefetch files.
recycle_bin: Parser for Windows $Recycle.Bin $I files.

skydrive_log: Parser for OneDrive (or SkyDrive) log files.
skydrive_log_error: Parser for OneDrive (or SkyDrive) error log files.
sqlite: Parser for SQLite database files.
symantec_scanlog: Parser for Symantec Anti-Virus log files.
winevt: Parser for Windows EventLog (EVT) files.
winevtx: Parser for Windows XML EventLog (EVTX) files.
winfirewall: Parser for Windows Firewall Log files.
winiis: Parser for Microsoft IIS log files.
winjob: Parser for Windows Scheduled Task job (or At-job) files.
winreg: Parser for Windows NT Registry (REGF) files.
google_drive: Parser for Google Drive SQLite database files.
skype: Parser for Skype SQLite database files.

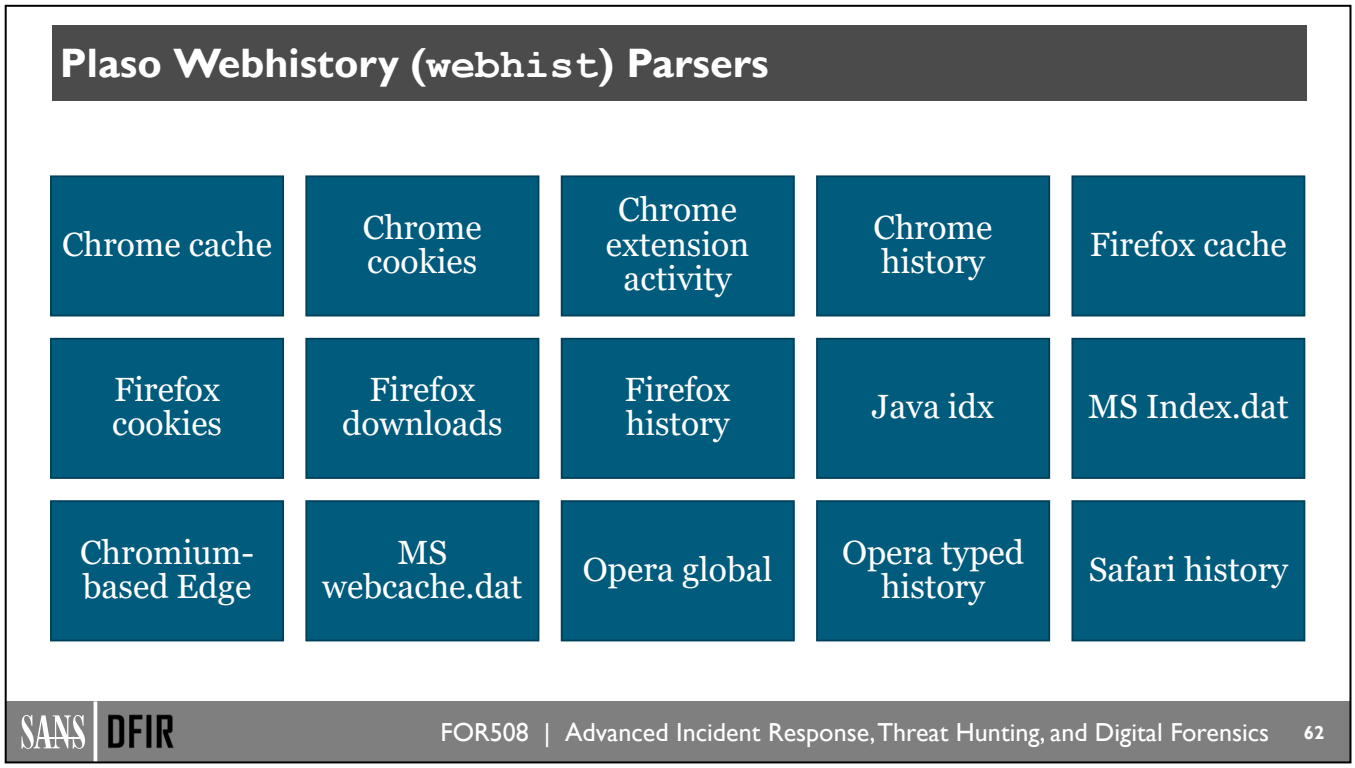


One of the areas that Log2timeline/Plaso has excelled in is extraction from Windows registry hive files. Some of the most evidentiary rich artifacts are found in the Windows Registry files, especially the NTUSER.DAT hive. Shellbag folder history, RecentDocs, and Open/SaveMRU lists account for much of the evidence of file opening or folder opening that a user accomplished. Recent additions in the evidence of execution category include appcompatcache, userassist, runmru, and others.

The combination of so many registry parsers/plugins makes Log2timeline/Plaso a near perfect forensic tool to run on Windows operating systems.

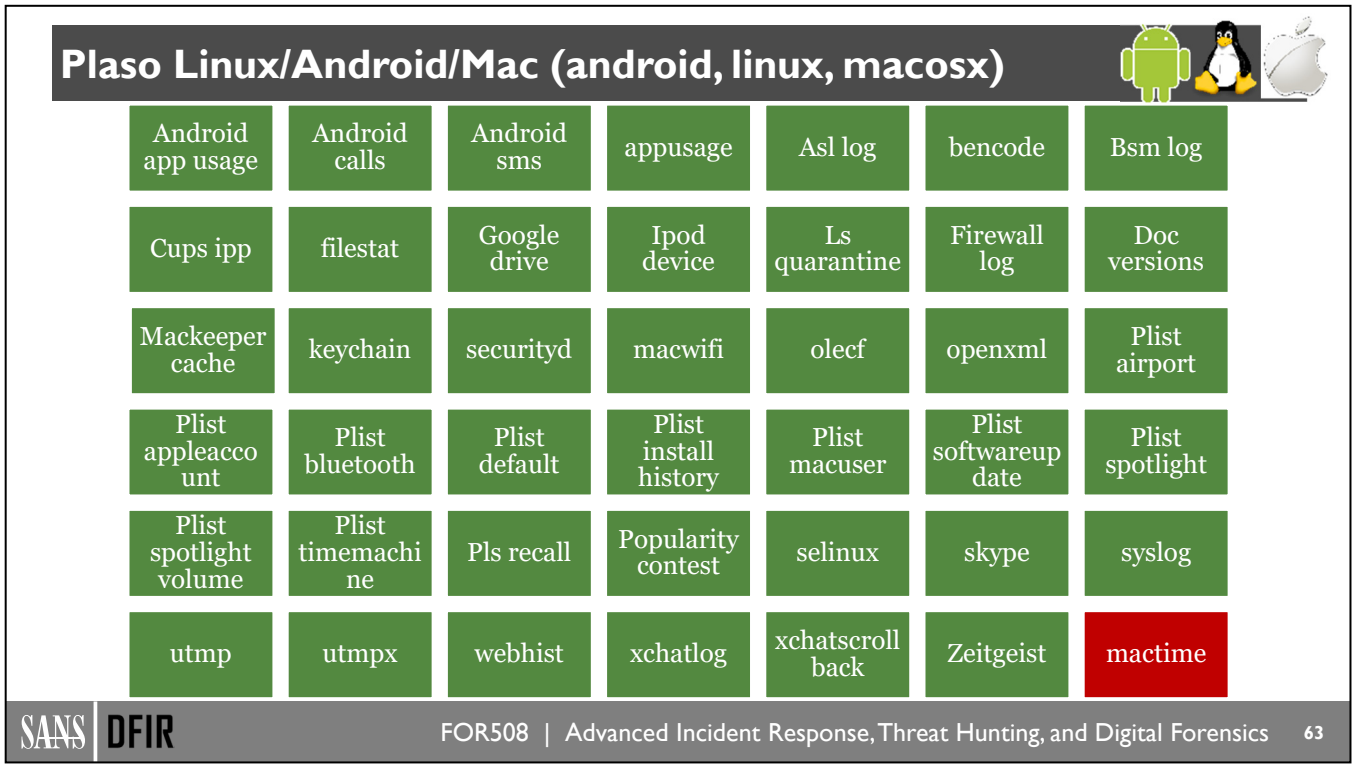
Registry Artifact Parsers
winreg_appcompatcache: Parser for Application Compatibility Cache Registry data
winreg_bagmru: Parser for BagMRU Registry data
winreg_boot_execute: Parser for Boot Execution Registry data
winreg_boot_verify: Parser for Boot Verification Registry data
winreg_ccleaner: Parser for CCleaner Registry data
winreg_default: Parser for Registry data
winreg_mountpoints2: Parser for mount points Registry data
winreg_mrulist_shell_item_list: Parser for Most Recently Used (MRU) Registry data
winreg_mrulist_string: Parser for Most Recently Used (MRU) Registry data
winreg_mrulistex_shell_item_list: Parser for Most Recently Used (MRU) Registry data
olecf: Parser for OLE Compound Files (OLECF).
olecf_automatic_destinations: Parser for *.automaticDestinations-ms OLECF files.

olecf_default: Parser for a generic OLECF item.
winreg_mrulistex_string: Parser for Most Recently Used (MRU) Registry data
winreg_mrulistex_string_and_shell_item: Parser for Most Recently Used (MRU) Registry data
winreg_mrulistex_string_and_shell_item_list: Parser for Most Recently Used (MRU) Registry data
winreg_msie_zone: Parser for Internet Explorer zone settings Registry data
winreg_msie_zone_software: Parser for Internet Explorer zone settings Registry data
winreg_office_mru: Parser for Microsoft Office MRU Registry data
winreg_outlook_mru: Parser for Microsoft Outlook search MRU Registry data
winreg_rdp: Parser for Terminal Server Client Connection Registry data
winreg_rdp_mru: Parser for Terminal Server Client MRU Registry data
winreg_run: Parser for run and run once Registry data
winreg_run_software: Parser for run and run once Registry data
winreg_sam_users: Parser for SAM Users and Names Registry keys
winreg_services: Parser for services and drivers Registry data
winreg_shutdown: Parser for ShutdownTime Registry value
winreg_task_cache: Parser for Task Scheduler cache Registry data
winreg_typed_urls: Parser for Internet Explorer typed URLs Registry data
winreg_usb: Parser for USB storage Registry data
winreg_usbstor: Parser for USB storage Registry data
winreg_userassist: Parser for User Assist Registry data
winreg_winrar: Parser for WinRAR History Registry data
winreg_winver: Parser for Windows version Registry data



Plaso/Log2timeline.py also has many new web history parsers in it. From older IE6-9 index.dat and cache indexes, to the IE10/11 webcachev01.dat ESEDB files, to SQLite databases from Chrome, Firefox, Safari, and Opera. Even Java IDX files, the index file associated with JAVA downloads, can be extracted and added to the timeline.

Webhistory Parsers
chrome_cache: Parser for Chrome Cache files
chrome_cookies: Parser for Chrome cookies SQLite database files
chrome_extension_activity: Parser for Chrome extension activity SQLite database files
chrome_history: Parser for Chrome history SQLite database files
firefox_cache: Parser for Firefox Cache files
firefox_cookies: Parser for Firefox cookies SQLite database files
firefox_downloads: Parser for Firefox downloads SQLite database files
firefox_history: Parser for Firefox history SQLite database files
java_idx: Parser for Java IDX files
msiecf: Parser for MSIE Cache Files (MSIECF) also known as index.dat
msie_webcache: Parser for MSIE WebCache ESE database files
opera_global: Parser for Opera global_history.dat files
opera_typed_history: Parser for Opera typed_history.xml files



The new version of log2timeline.py/plaso has many new parsers in it, expanding its reach onto other platforms.

Linux / Android /Other
android_app_usage: Parser for the Android usage-history.xml file
android_calls: Parser for Android calls SQLite database files
android_sms: Parser for Android text messages SQLite database files
bencode: Parser for bencoded files
bencode_transmission: Parser for Transmission bencoded files
bencode_utorrent: Parser for uTorrent bencoded files
filestat: Parser for filesystem stat information
google_drive: Parser for Google Drive SQLite database files
openxml: Parser for OpenXML (OXML) files
pls_recall: Parser for PL-SQL Recall files
popularity_contest: Parser for popularity contest log files
selinux: Parser for SELinux audit log files
syslog: Parser for syslog files
skype: Parser for Skype SQLite database files
utmp: Parser for Linux/Unix UTMP files
utmpx: Parser for UTMPX files
xchatlog: Parser for XChat log files
xchatscrollback: Parser for XChat scrollback log files
zeitgeist: Parser for Zeitgeist activity SQLite database files
mactime: Parser for SleuthKit's mactime bodyfiles
pcap: Parser for PCAP files

Mac
appusage: Parser for Mac OS X application usage SQLite database files
asl_log: Parser for ASL log files
bencode: Parser for bencoded files
bencode_transmission: Parser for Transmission bencoded files
bencode_utorrent: Parser for uTorrent bencoded files
bsm_log: Parser for BSM log files
cups_ipp: Parser for CUPS IPP files
filestat: Parser for filesystem stat information
google_drive: Parser for Google Drive SQLite database files
ipod_device: Parser for iPod, iPad and iPhone plist files
ls_quarantine: Parser for LS quarantine events SQLite database files
mac_appfirewall_log: Parser for appfirewall.log files
mac_keychain: Parser for Mac OS X Keychain files
mac_securityd: Parser for Mac OS X securityd log files
macwifi: Parser for Mac OS X wifi.log files
mac_document_versions: Parser for document revisions SQLite database files
mackeeper_cache: Parser for MacKeeper Cache SQLite database files
olecf: Parser for OLE Compound Files (OLECF)
olecf_automatic_destinations: Parser for *.automaticDestinations-ms OLECF files
olecf_default: Parser for a generic OLECF item
olecf_document_summary: Parser for a DocumentSummaryInformation OLECF stream
olecf_summary: Parser for a SummaryInformation OLECF stream
openxml: Parser for OpenXML (OXML) files
plist_airport: Parser for Airport plist files
plist_appleaccount: Parser for Apple account information plist files
plist_bluetooth: Parser for Bluetooth plist files
plist_default: Parser for plist files
plist_install_history: Parser for installation history plist files
plist_macuser: Parser for Mac OS X user plist files
plist_softwareupdate: Parser for Mac OS X software update plist files
plist_spotlight: Parser for Spotlight plist files
plist_spotlight_volume: Parser for Spotlight volume configuration plist files
plist_timemachine: Parser for TimeMachine plist files
skype: Parser for Skype SQLite database files
webhist: All the Web History files

Other
mactime: Parser for common mactime (SleuthKit) bodyfile format

The mactime plugin converts traditional “body” file timeline output into a plaso database format file. This allows tools that default to mactime format (such as the timeliner Volatility plugin) to be imported directly into a plaso.dump database. The following command runs the parser “mactime” against a bodyfile and adds it directly into a plaso.dump database. If the plaso.dump database already exists, it will append to the database.

```
log2timeline.py --parsers “mactime” plaso.dump bodyfile
```


log2timeline.py Usage

```
# log2timeline.py --storage-file [STORAGE FILE] [SOURCE]
```

Important Options:

--storage-file	Plaso output database file - **/path/to/output.dump**
[SOURCE]	Device, image, or directory of files to be parsed **/path/to/image.dd**
-z *<TIMEZONE>*	Define the time zone of the system being investigated (not the output). If a forensic image is provided (e.g. E01, raw), the timezone will be identified automatically
--zone list	List available time zones
--help	List all options with usage descriptions

Because Plaso is really a backend parsing engine, it needs a frontend to be able to run as a standalone tool. There are many possible frontends. The primary one is called log2timeline, which is the single-machine frontend that takes care of processing data from files in a directory, mounted devices, forensic images, and virtual disk images. When run against a collection of files in a directory, recursion is automatic, evaluating all files through all subdirectories.

Given the power of the tool, running log2timeline can be quite simple. However, a few options need to be understood for accurate parsing. We'll also discuss additional options in the coming pages to help optimize processing time. You can always see a full list of options by running "log2timeline.py --help".

The two required options are (1) specifying a storage file and (2) specifying the source of data to parse. The storage file is a database file that holds normalized parsed data resulting from log2timeline analysis of artifacts. This is typically a new file created as a result of running log2timeline, but it's also possible to re-run log2timeline against additional data to add events to an existing database file. The source of data to parse is a directory of files, a mount point, or an image file containing artifact files from the subject system.

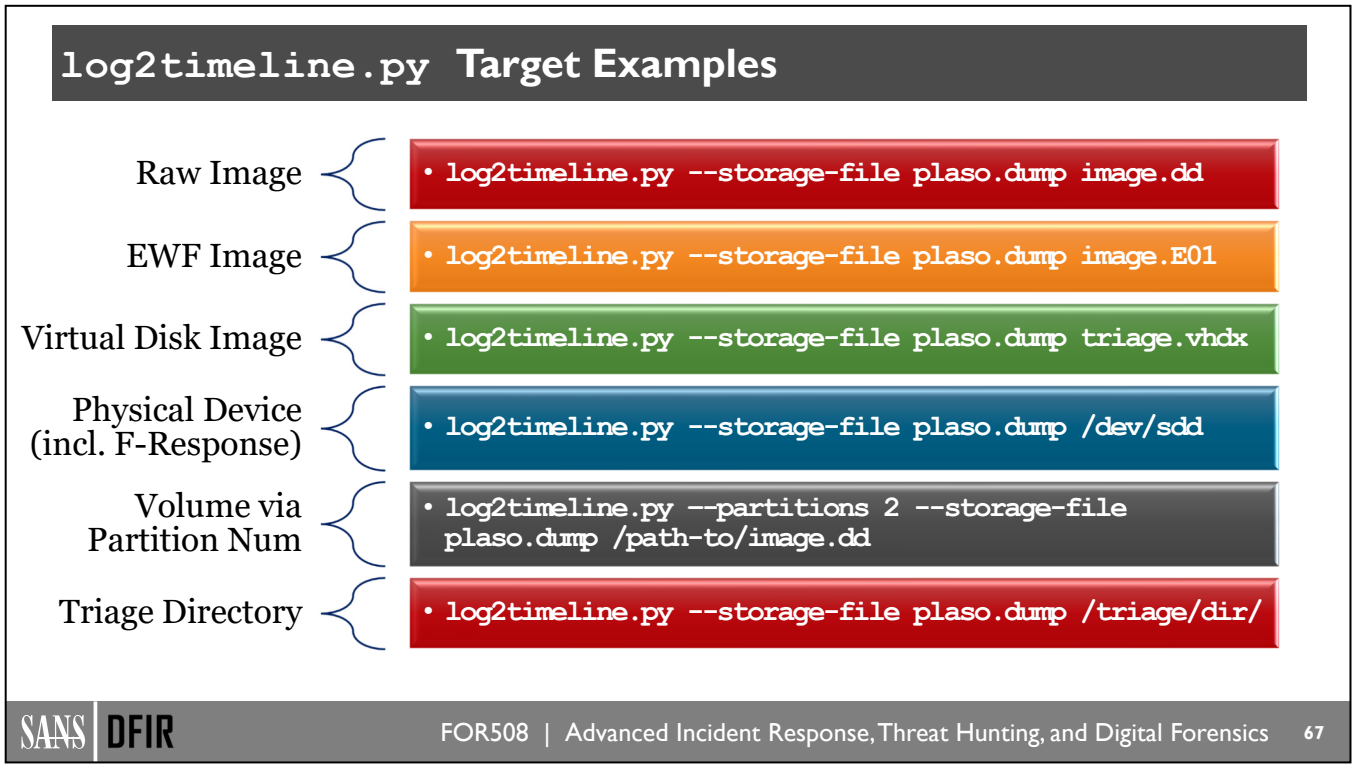
Dealing with time zones is a painful but important part of forensic analysis and reporting. As such, we need to understand how our forensic tools deal with them. In the case of Plaso, there are options for handling time zones during the input stage and the output stage. Log2timeline is the input stage, and so the option "-z TIMEZONE" is important to understand. This is the time zone of the computer being investigated (not the desired output time zone). This is needed because some artifacts are stored in local time instead of UTC. Log2timeline is aware of this, but it needs to know how much to adjust the local times for certain artifacts to be consistent with the other artifacts it's collecting that are stored in UTC. Note that if a forensic image or mounted device is used as the source, log2timeline typically runs a preprocessor that collects the time zone information automatically, and if discovered, that value will be used where applicable. When run against a collection of files, including a triage image, the time zone should be included. Best practice is to not take chances and specify the time zone of the subject system, especially if you know there are artifacts in the source

files being parsed that store times in local time. If time zone is not specified, and not detected, UTC is used for the artifacts that are stored in local time. (Log2timeline always stores UTC-based artifacts in UTC.) To get a list of available time zones, run "log2timeline.py --zone list".

A pro tip for validating proper local time adjustments for Windows hosts is to review the "setupapi" parsing results. This parser is included in log2timeline's Windows presets and is one of just a handful of Windows artifacts that store times in local time. The good news for Windows analysis is that the vast majority of artifacts are stored in UTC by default.

Plaso is a powerful framework with a lot of options and features. For more details on the project, be sure to visit the Plaso documentation site.[1]

[1] https://for508.com/j0cqb



Log2timeline is an amazingly flexible tool. It can run against nearly any data source you have available. It supports forensic images (raw "dd" and E01), virtual disk images (VMDK, VHD/X, and QCOW/2/3), mounted devices (e.g., hard drive attached via a write-blocking device), individual volumes on a drive, and one of our favorite use cases, parsing a collection of triage data. Imagine you only had a handful of event logs, the prefetch folder, and a browser database exported from a subject system. You could point log2timeline at that collection of files and create a timeline out of whatever artifacts you happen to have available. Awesome!

The following are examples of log2timeline parsing evidence in different formats:

Raw Image

```
log2timeline.py --storage-file /path-to/plaso.dump /path-to/image.dd
```

EWF Image

```
log2timeline.py --storage-file /path-to/plaso.dump /path-to/image.E01
```

Virtual Disk Image

```
log2timeline.py --storage-file /path-to/plaso.dump /path-to/triage.vhdx
```

Physical Device (e.g., attached mounted drive, write-blocked drive, or remotely connected F-Response drive)

```
log2timeline.py --storage-file /path-to/plaso.dump /dev/sdd
```

Volume via Partition Number(s) (i.e., from a full disk image, one or more partitions can be specified)

```
log2timeline.py --partitions 2 --storage-file /path-to/plaso.dump image.dd
```

Triage Folder

```
log2timeline.py --storage-file /path-to/plaso.dump /triage/directory/
```

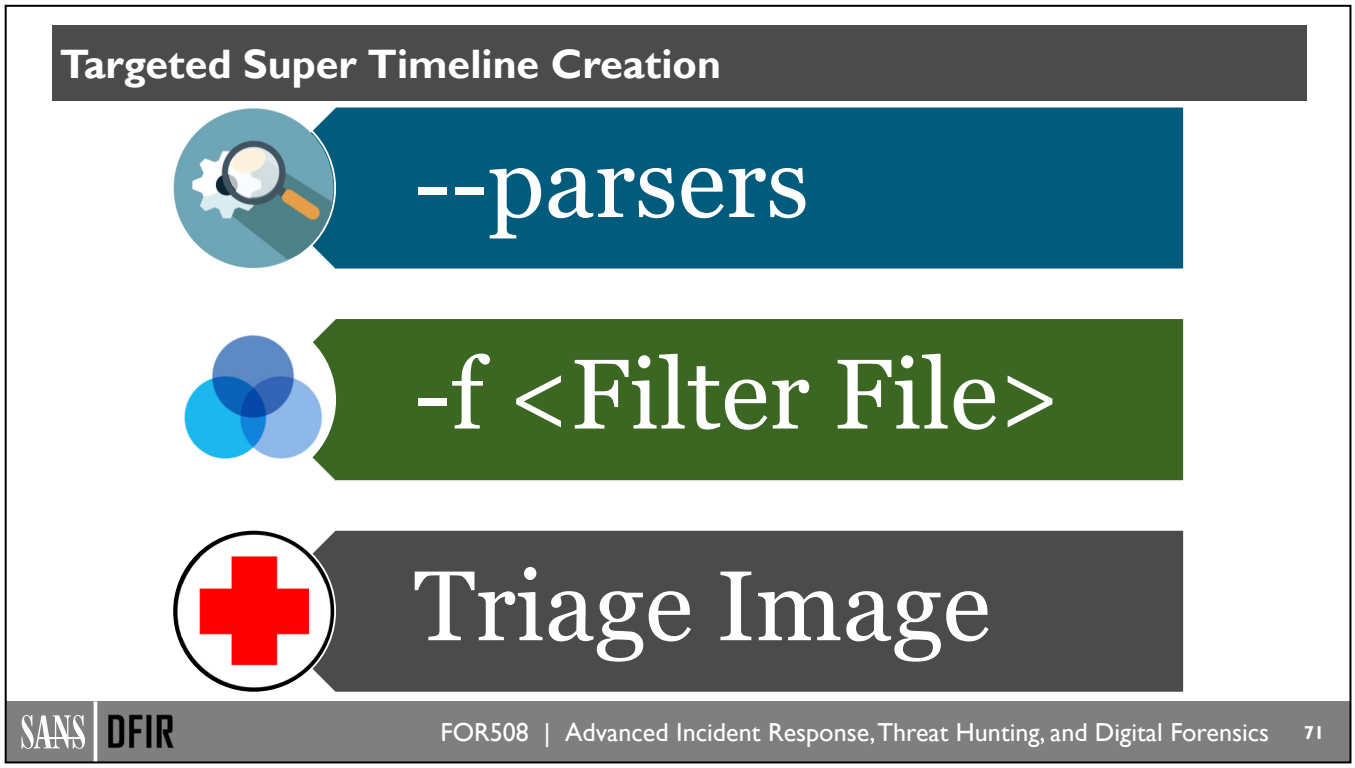
Note that when processing a disk image, it's recommended to parse the image file directly rather than mount the image to a mount point on the analysis system. There are several drawbacks of using a mount point, such as differences in which files are exposed depending on the driver used to mount the disk image. For example, on an NTFS image, the $MFT master file table may not be exposed for log2timeline to parse. Therefore, it's generally better to point log2timeline at the full disk image when an image is available. Note that image parsing is provided via the Digital Forensics Virtual File System (dfVFS) toolset. Visit the dfVFS documentation site for details on the latest storage media file types supported.[1]

[1] dfVFS documentation: https://for508.com/8vhmg

This page intentionally left blank.

Targeted Super Timeline Creation

This page intentionally left blank.



In the world of timeline analysis, analysts tend to split into two schools of thought: those who prefer an all-inclusive super timeline and those that lean toward more targeted mini-timelines.

The all-inclusive or "kitchen sink" approach runs log2timeline against a disk image and extracts every timestamp and artifact it supports. Filtering and analysis are then accomplished on the full dataset after collection. This is what many people think of when they hear “super timeline.” Smaller timelines can always be carved out of the larger set when necessary.

The second school of thought uses a more targeted acquisition of data, collecting data only from files relevant for their needs. These analysts tend towards the more advanced features of log2timeline in combination with other tools, such as one-off scripts and dedicated parsing tools. The advantage of targeted collection is speed. An all-inclusive run of log2timeline against a large disk could easily take 12-24 hours to complete. A targeted collection often requires between 5-30 minutes. The disadvantage of creating smaller targeted timelines is the possibility of missing an important artifact. However, for most cases starting your analysis early is worth the small risk, especially knowing you can usually go back and add artifacts as needed once you have a better understanding of the case and data available. Log2timeline.py now ships with multiple features to facilitate targeted timeline creation.

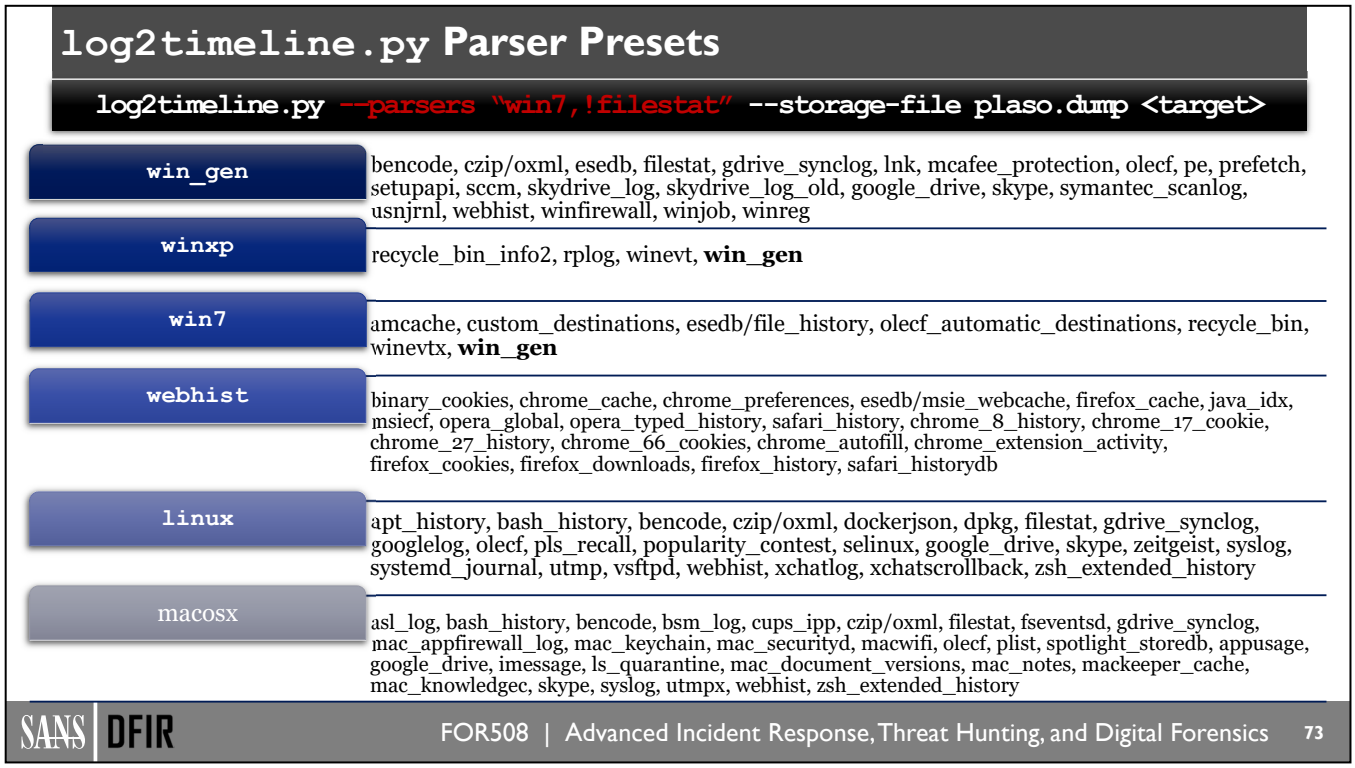
Parsers
`--parsers PARSER_LIST`

Log2timeline makes it easy to provide a specific list of parsers to use for timeline creation. This is a straightforward way to limit scope, as only artifacts relevant to the chosen parsers will be analyzed and output. Parsers can be specified by name directly on the command line using the --parsers option. To see the full list of available parsers, run log2timeline with the “--parsers list” option.

Filter Files
`-f FILTER_FILE, --file_filter FILTER_FILE, --file-filter FILTER_FILE`

Instead of limiting artifact analysis by parser type, a filter file contains a list of specific files to find and parse. A filter file contains one file path per line. While it might seem tedious to specify every file to parse, in practice it turns out to be a relatively small list, aligning almost exactly with what one might collect with a triage image of a system. And as we will see shortly, SANS has already created an excellent list for your use. This feature is particularly fast and useful when you have a full disk to analyze and do not want to take the extra step of creating a triage file collection first.

Using Triage Images
While technically not a feature of log2timeline, one of the easiest ways to reduce the scope of your timeline is to point at a triage file collection instead of the entire disk. By definition, triage collection collects the files you believe to be most vital to your investigation. Hence, it makes sense to create a timeline from those files and it takes no additional work other than pointing log2timeline at the correct data source. This is the technique we will employ in an upcoming exercise.



Defining a list of parsers to use is the original way to speed up processing in log2timeline. Log2timeline now provides several "preset" lists in the distribution. On the command-line, presets can be referenced in addition to individual parsers using a comma-separated list. Parsers can be added to or excluded from the presets. For example, a parser can be excluded from a preset by prepending the parser with an exclamation mark (!). In the example on this slide, the "win7" preset list of parsers will be used, with the exception of the individual "filestat" parser.

The slide also shows the list of parser presets available in log2timeline at the time of this writing. Perhaps somewhat confusingly, the "win7" preset is relevant for all Windows systems from Vista and above (many artifacts changed formats between XP and Windows Vista). Note that both "win7" and "winxp" reference "win_gen", which means they include that additional list of parsers, which are generally consistent across all Windows versions.

To get the list of available parsers and presets on a given installation of log2timeline, use the "--parsers list" option. You can also learn more about parsers, plugins, and presets from the Plaso's developer documentation.[1][2]

[1] How to write a parser: https://for508.com/i9k8v
[2] Parsers and plugins: https://for508.com/s0zjl

log2timeline.py Filter Files

log2timeline.py -f filter_windows.txt

```
# Windows Registry files.
/(Users|Documents And Settings)/.+/NTUSER[.]DAT
/Users/.+/AppData/Local/Microsoft/Windows/Usrclass[.]dat
/Documents And Settings/.+/Local Settings/Application Data/Microsoft/Windows/Usrclass[.]dat
{systemroot}/System32/config/(SAM|SOFTWARE|SECURITY|SYSTEM)
# Recent file activity.
/Users/.+/AppData/Roaming/Microsoft/Windows/Recent/.+[.]lnk
/Users/.+/AppData/Roaming/Microsoft/Office/Recent/.+[.]lnk
/Documents And Settings/.+/Recent/.+[.]lnk
# Jump List files.
/Users/.+/AppData/Roaming/Microsoft/Windows/Recent/Automaticdestinations/.+[.]automaticDestinations-ms
/Users/.+/AppData/Roaming/Microsoft/Windows/Recent/Customdestinations/.+[.].customDestinations-ms
# Windows Event Logs.
{systemroot}/System32/winevt/Logs/.+[.]evtx
{systemroot}/System32/config/.+[.]evt
# Various log files.
{systemroot}/inf/setupapi[.].+[.]log
{systemroot}/setupapi.log
```

- Filter files allow for targeted analysis within large data sets
- Great for processing disk images
- Two formats supported:
  - Legacy text-based
  - YAML-based

- Both formats support:
  - Basic regular expressions
  - Wildcards
  - Path recursion
  - Path variables
- Only the YAML format supports exclusion filters

log2timeline.py -f filter_windows.yaml

```
---
description: Windows Registry files.
type: include
path_separator: '\'
paths:
- '\\(Users|Documents And Settings)\\.+\\NTUSER[.]DAT'
- '\\Users\\.+\\AppData\\Local\\Microsoft\\Windows\\Usrclass[.]dat'
- '\\Documents And Settings\\.+\\Local Settings\\Application Data\\Microsoft\\Windows\\Usrclass[.]dat'
- '%SystemRoot%\\System32\\config\\(SAM|SOFTWARE|SECURITY|SYSTEM)'
---
description: Recent activity files.
type: include
path_separator: '\'
paths:
- '\\Users\\.+\\AppData\\Roaming\\Microsoft\\Windows\\Recent\\.+[.]lnk'
- '\\Users\\.+\\AppData\\Roaming\\Microsoft\\Office\\Recent\\.+[.]lnk'
- '\\Documents And Settings\\.+\\Recent\\.+[.]lnk'
---
```

SANS DFIR FOR508 | Advanced Incident Response, Threat Hunting, and Digital Forensics 74

Filter files in log2timeline work by limiting the targeted set of input. The file defines the paths to files and paths that should be parsed, ignoring the rest. This opens up the possibility of a fast method of extracting just the data of interest, regardless of the data source. Filter files allow log2timeline to skip a majority of irrelevant files and folders present in a filesystem. This speeds up processing by orders of magnitude and should be considered in any situation in which an analyst quickly needs to answer questions about a system.

There are two supported formats for the filter file: the traditional text-based format and the newer YAML-based format.

Text-based Filter Files

The text-based format contains a single line for each file path with each part of the path separated by a forward slash. Entries can use regular expressions and wildcards to allow for both precise and broader targeting. "Path expansion variables" extracted from Plaso preprocessing modules are also allowed (for example, the variable {systemroot} shown in the slide).

How is a filter file built? Let's start with an example filter entry:

```
/(Users|Documents And Settings)/.+/NTUSER.DAT
```

This regular expression states we are looking for a file within the folder "Users" or "Documents and Settings" followed by any subfolder (.+) which includes a file specifically named NTUSER.DAT. In other words, this line defines that Plaso should parse the NTUSER.DAT file for every user profile on the system.

Let's try another one:

```
{systemroot}/winevt/Logs/.+evtx
```

This line tells Plaso to parse all EVTX files stored under the "winevt\Logs" path in the system root directory.

Note that this line contains a path expansion variable of {systemroot}. These variables are defined by curly braces { } and they will be replaced by a path value discovered during preprocessing of a disk image. Be aware, however, experience shows that the mapping of these variables is not 100% effective. Although it's a nice feature in theory, in practice it's often better to specify the entire path rather than rely on variable discovery.

Keep in mind that all expressions in the file are case-insensitive. Special characters that should be treated as literals should be surrounded by square brackets, for example [$]. Finally, it's important to note that the text-based filters only define what to include. If a file or path is not defined in the filter file, it will be ignored by log2timeline.

YAML-based Filter Files
YAML filter files were introduced to provide for a little more flexibility in defining filtering rules. The major difference in capability is that YAML-based filtering supports exclusion rules. As such, rules can be created to define not only what to include, but also what not to include.

For example, let's say we wanted to do the opposite of what we did in the last example. I.e., we want to *exclude* all event logs. The following YAML definition would accomplish this task:

```
description: Exclude Windows Event Log files
type: exclude
path_separator: '\'
paths:
- '%SystemRoot%\\System32\\config\\.+[.]evt'
```

In this example, we see typical YAML syntax. The important directive of "type: exclude" makes it clear that any files or paths which match this rule should not be processed by log2timeline. The path regex looks familiar, as well as the path expansion variable, except now the variable is surrounded by percent signs rather than curly braces.

Overall, filter files provide analysts with significant flexibility in limiting the scope of log2timeline processing. This has the dual benefit of saving valuable time and computing resources. In testing, it has reduced processing time of full disk images by 90% on average. Employing filter files and/or limiting parsers is an important capability that can greatly enhance a responder's speed and effectiveness.

Reference the Plaso documentation for the latest information about filter files.[1] The Plaso team maintains a couple of sample filter files for Windows in their GitHub repository (one text-based and one YAML-based).[2]

[1] Filter file documentation: https://for508.com/aws2z
[2] Filter files from Plaso project: https://for508.com/n0qg3

Fast Forensics/Triage Extraction

Artifact	Course Covered
Memory	• FOR500/508
Registry Hives and Backups	• FOR500
LNK Files	• FOR500
Jump Lists	• FOR500
Prefetch	• FOR500/508
Event Logs and Windows Logs	• FOR500/508
Browser Data (IE, Firefox, Chrome)	• FOR500
Master File Table ($MFT)	• FOR508
Log Files and Journal Log	• FOR508
Pagefile and Hibernation Files	• FOR508

With today's large-scale intrusions and massive amounts of data, we can no longer expect to image every system of interest. A full physical image (and memory image) is still the "gold standard" because it gives us the greatest number of analysis options, but it simply isn't feasible in every situation. Preparing for this situation, if you can't do a full disk image, what files should you consider collecting from a system to perform quick analysis and triage?

The set of artifacts listed on this slide enables memory forensics, full registry analysis, application execution analysis, taking advantage of any relevant event logging, and performing a good timeline analysis (though a full super timeline analysis would require an image of the volume). A majority of the questions most analysts need to answer could be answered by this limited set of data.

Depending on what you are investigating, there are many other items you might consider collecting: internet browser database files, cloud storage databases, email archives, and even the entire user profile for individuals of interest. There is no right or wrong answer. Fast forensics is limited only by the time allotted and the knowledge and creativity of the incident responder.

Triage Image Timelining


```
log2timeline.py --storage-file plaso.dump /triage-output/
```


Filter files and specifying parsers are excellent ways to speed up log2timeline and speed up your time to investigate. However, the easiest option could be to just limit the set of data you use as input. If you have a collection of triage files, there is no need to limit your timeline collection further. By definition, triage collection collects the files you believe to be most vital to your investigation. Hence, it makes sense to create a timeline from those files and it takes no additional work other than pointing log2timeline at the correct data source. Thus, a combination of a tool like KAPE for triage collection and Plaso/log2timeline for timeline creation creates a rapid forensics capability, allowing forensic collection and analysis to be conducted in minutes.[1] The choice of triage tool to use is not important to this process. The most important takeaway is log2timeline can create a timeline out of whatever collection of files you have available (KAPE just happens to be a very easy and powerful way to collect files). This is a radical departure from traditional disk-based forensics in that it does not require full acquisition of a massive hard drive. However, this capability does not need to replace your existing processes. A triage image can be collected and processed while a follow-up action acquires full disk images of the target systems. If you do this a few times, our bet is you will find very few reasons to go back to full disk imaging as your standard operating procedure!

[1] KAPE Documentation: https://for508.com/l5o3e

Timeline Analysis Agenda

- Timeline Analysis Overview
- Filesystem Timeline Creation and Analysis
- Introducing the Super Timeline
- Targeted Super Timeline Creation
- Filtering the Super Timeline
- Super Timeline Analysis

This page intentionally left blank.

Filtering the Super Timeline

Using `pinfo.py` and `psort.py`

This page intentionally left blank.

Plaso Database Information : `pinfo.py`

```
pinfo.py –v plaso.dump
```

- **`pinfo.py`** displays contents of Plaso database
  - **`-v`** for "verbose" information
- Information stored inside the **`plaso.dump`** storage container:
  - Information on when and how the tool was run
  - List of all plugins/parsers used
  - Filter file information (if applicable)
  - Information gathered during the preprocessing stage
  - A count of each artifact parsed
  - Errors and storage container metadata

Pinfo.py is an important tool to validate the plaso database file. It will list the internal metadata contained in the database to identify what was parsed, when the parsing took place, any preprocessor information, and the plugins used to extract data. The -v option is recommended and produces verbose output including preprocessor data.

```
pinfo.py –v plaso.dump | less
```

```
************************** Plaso Storage Information **************************
         Filename : plaso.dump
    Format version : 20210105
      Storage type : session
Serialization format : json
---------------------------------------------------------------------------

********************************** Sessions ***********************************
693a26e0-84e0-4bff-b24e-f81301f5a5a8 : 2021-05-16T04:41:09.413962Z
a5001f19-62b4-44cb-ad03-8da4964867ab : 2021-05-16T05:03:28.645566Z
---------------------------------------------------------------------------

**************** Session: 693a26e0-84e0-4bff-b24e-f81301f5a5a8 ****************
            Start time : 2021-05-16T04:41:09.413962Z
        Completion time : 2021-05-16T05:03:05.278313Z
```

```
 Product name : plaso
        Product version : 20210412
   Command line arguments : /usr/bin/log2timeline.py -z EST5EDT --parsers
                            win7,!filestat /cases/EDR-triage/plaso.dump
                            /mnt/baserd01-triage/C/
  Parser filter expression : win7,!filestat
Enabled parser and plugins : !filestat, amcache, bencode, binary_cookies,
                            chrome_cache, chrome_preferences,
                            custom_destinations, czip/oxml, esedb,
                            esedb/file_history, esedb/msie_webcache,
                            firefox_cache, gdrive_synclog, java_idx, lnk,
                            mcafee_protection, msiecf, olecf,
                            olecf/olecf_automatic_destinations, opera_global,
                            opera_typed_history, pe, plist/safari_history,
                            prefetch, recycle_bin, sccm, setupapi,
                            skydrive_log, skydrive_log_old,
                            sqlite/chrome_17_cookies,
                            sqlite/chrome_27_history,
                            sqlite/chrome_66_cookies, sqlite/chrome_8_history,
                            sqlite/chrome_autofill,
                            sqlite/chrome_extension_activity,
                            sqlite/firefox_cookies, sqlite/firefox_downloads,
                            sqlite/firefox_history, sqlite/google_drive,
                            sqlite/safari_historydb, sqlite/skype,
                            symantec_scanlog, usnjrnl, winevtx, winfirewall,
                            winjob, winreg
       Preferred encoding : UTF-8
             Debug mode : False
         Artifact filters : N/A
            Filter file : N/A
---------------------------------------------------------------------------

********** System configuration: 693a26e0-84e0-4bff-b24e-f81301f5a5a8 **********
              Hostname : BASE-RD-01
       Operating system : Windows NT
Operating system product : Windows 10 Enterprise
Operating system version : 6.3
            Code page : cp1252
        Keyboard layout : N/A
             Time zone : EST5EDT
---------------------------------------------------------------------------
```

********** User accounts: 693a26e0-84e0-4bff-b24e-f81301f5a5a8 **********
Username : User directory
---
systemprofile : %systemroot%\system32\config\systemprofile
LocalService : C:\WINDOWS\ServiceProfiles\LocalService
NetworkService : C:\WINDOWS\ServiceProfiles\NetworkService
tdungan : C:\Users\tdungan
rsydow-a : C:\Users\rsydow-a
cbarton-a : C:\Users\cbarton-a
spsql : C:\Users\spsql
administrator.shieldbase : C:\Users\administrator.shieldbase

****** Events generated per parser: 693a26e0-84e0-4bff-b24e-f81301f5a5a8 *******
Parser (plugin) name : Number of events
---
appcompatcache : 796
bagmru : 250
bam : 57
chrome_27_history : 720
chrome_66_cookies : 4622
chrome_autofill : 185
chrome_cache : 5829
chrome_preferences : 40
explorer_mountpoints2 : 12
explorer_programscache : 3
firefox_cookies : 1434
firefox_history : 88
google_analytics_utma : 11
google_analytics_utmb : 5
google_analytics_utmt : 4
google_analytics_utmz : 9
lnk : 1435
mrulist_string : 25
mrulistex_shell_item_list : 10
mrulistex_string : 27
mrulistex_string_and_shell_item : 25
mrulistex_string_and_shell_item_list : 1
msie_webcache : 3712
msie_zone : 72
mstsc_rdp : 6
mstsc_rdp_mru : 1


```
                network_drives : 3
                      networks : 4
      olecf_automatic_destinations : 484
                  olecf_default : 44
                          oxml : 94
                            pe : 4293
                      prefetch : 1286
                   recycle_bin : 1
                      setupapi : 544
                   shell_items : 2304
                   userassist : 121
             windows_boot_execute : 2
                    windows_run : 16
                windows_sam_users : 12
                 windows_services : 625
                 windows_shutdown : 2
               windows_task_cache : 587
                 windows_timezone : 1
                windows_typed_urls : 9
                  windows_version : 4
                        winevtx : 557607
                      winlogon : 4
                  winreg_default : 405292
                        Total : 992718
-------------------------------------------------------------------------------

**************** Session: a5001f19-62b4-44cb-ad03-8da4964867ab ****************
            Start time : 2021-05-16T05:03:28.645566Z
        Completion time : 2021-05-16T05:08:27.332783Z
          Product name : plaso
        Product version : 20210412
   Command line arguments : /usr/bin/log2timeline.py -z EST5EDT --parsers
                        mactime /cases/EDR-triage/plaso.dump
                        /cases/precooked/timeline/baserd01-mftecmd.body
  Parser filter expression : mactime
Enabled parser and plugins : mactime
      Preferred encoding : UTF-8
            Debug mode : False
        Artifact filters : N/A
            Filter file : N/A
-------------------------------------------------------------------------------
```

```
********** System configuration: a5001f19-62b4-44cb-ad03-8da4964867ab **********
             Hostname : N/A
       Operating system : N/A
Operating system product : N/A
Operating system version : N/A
            Code page : cp1252
       Keyboard layout : N/A
            Time zone : EST5EDT
--------------------------------------------------------------------------------

********** Available time zones: a5001f19-62b4-44cb-ad03-8da4964867ab **********
Name :
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

************* User accounts: a5001f19-62b4-44cb-ad03-8da4964867ab **************
Username : User directory
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

****** Events generated per parser: a5001f19-62b4-44cb-ad03-8da4964867ab *******
Parser (plugin) name : Number of events
--------------------------------------------------------------------------------
          mactime : 1488556
            Total : 1488556
--------------------------------------------------------------------------------

<SNIP>
```

Timeline Filtering : psort.py

```
psort.py --output-time-zone 'UTC' -o l2tcsv -w supertimeline.csv plaso.dump FILTER
```

`--output-time-zone ZONE`	Converts stored times to the specified time zone
`-o FORMAT:`	Choose the output module (default is "dynamic" minimal CSV)
`l2tcsv`	Traditional CSV format used by log2timeline
`elastic`	Sends result into an Elasticsearch database
`-w FILE`	Name of the output file to be written
`FILTER`	Filters argument (e.g., provide a date range filter)

```
date > datetime('2018-08-23T00:00:00') AND
date < datetime('2018-09-07T00:00:00')
```

```
sansforensics@SANS-SIFT: ~
plaso - psort version 20210412
Storage file            : /mnt/g/timeline/plaso.dump
Processing time         : 00:10:40
Events:         Filtered        In time slice   Duplicates      MACB grouped    Total
                2192197         0               447             283222          2481274
Identifier              PID     Status          Memory          Events          Tags            Reports
Main                    76      exporting       171.2 MiB       289077 (0)      0 (0)           0 (0)
Processing completed.
```

85

psort is a command line tool to post-process the Plaso storage database. It provides filtering, sorting, and de-duplication of the contents of Plaso database files.

The raw output of **log2timeline** is an SQLite database containing serialized event objects, metadata, tags, reports, and various other extracted or derived information. **psort** is the post-processing tool used to read the log2timeline database file and extract events from it in human-readable format. It allows post-process filtering against a wide range of attributes (on this slide we show a simple data range filter). Importantly, it also removes duplicate entries from the output. This is particularly helpful when including artifacts like Volume Shadow Copies and backup registry hives. Think of **psort** as the tool that creates the timeline from a set of extracted data. While you can create a timeline of all the data (the default), you can also rapidly create mini-timelines using a range of filters, including time and artifact based. Building the plaso database takes time, but filtering the results is typically much faster.

General Format of Command
The generic options are:
psort.py [--output-time-zone ZONE] [-o FORMAT] [-w OUTPUTFILE] STORAGE_FILE FILTER

Modify the Timezone
psort uses UTC as its default timezone when outputting events. This can be controlled using the "--output-time-zone" parameter. To see a list of all supported timezones, run "psort.py --output-time-zone list".

Outputting Everything
The minimum number of inputs needed for this utility to run is the path to a Plaso storage file and an output file (-w). Given this, the utility will return all events from the storage file and use UTC as the default time zone to output events.

Using a Time Slice:

```
psort.py --slice '2018-08-30T20:00:00' -w slice.csv plaso.dump
```

The time slice can be a good option when investigating a specific pivot point into the system. Let's say you have extracted events and you then run **psort** with a filter to show only signs of application execution. When examining that limited dataset, you come across an execution of something you believe is odd or something requiring additional inspection. You can note the time and grab a "slice" from that point in time. What happens is the tool will extract all events occurring 5 minutes before and 5 after the provided timestamp, providing a mini-timeline for analysis. The default slice is 5 minutes before and after. That can be adjusted using the "--slice_size" option.

Other Output Formats

The default output format is "dynamic", which is a CSV format with a slimmed down number of fields. In FOR508, we generally prefer the "l2tcsv" format with several additional fields. There are a number of other output formats, including sending the data to an Elasticsearch database. Excellent for scaling analysis! To see a list of all supported output modules, run "psort.py -o list". Here are the modules at the time of this writing:

```
******************************** Output Modules ********************************
     Name : Description
------------------------------------------------------------------------------
   dynamic : Dynamic selection of fields for a separated value output format.
   elastic : Saves the events into an Elasticsearch database.
elastic_ts : Saves the events into an Elasticsearch database for use with Timesketch.
      json : Saves the events into a JSON format.
 json_line : Saves the events into a JSON line format.
       kml : Saves events with geography data into a KML format.
    l2tcsv : CSV format used by legacy log2timeline, with 17 fixed fields.
    l2ttln : Extended TLN 7 field | delimited output.
      null : Output module that does not output anything.
     rawpy : native (or "raw") Python output.
       tln : TLN 5 field | delimited output.
      xlsx : Excel Spreadsheet (XLSX) output
------------------------------------------------------------------------------
```

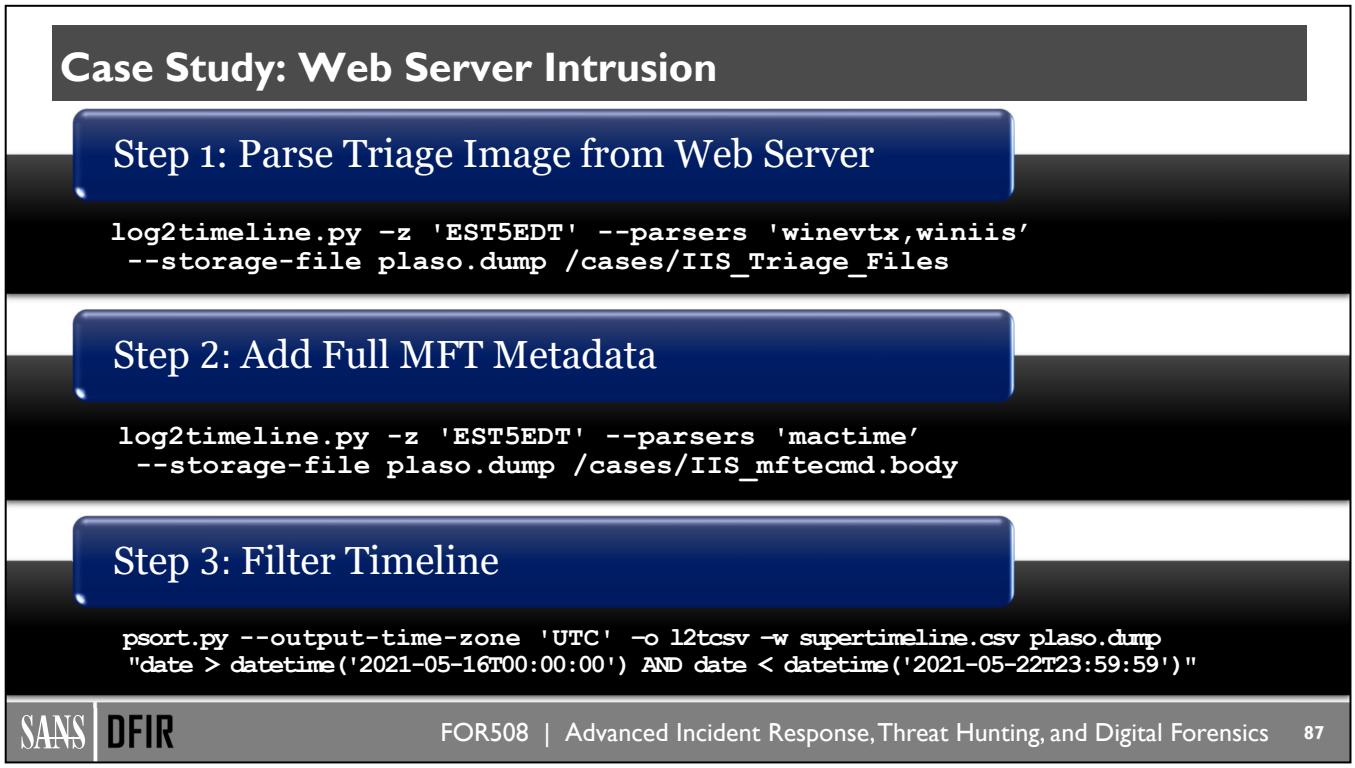
EXAMPLE

```
psort.py --output-time-zone 'UTC' -o l2tcsv -w plaso.csv plaso.dump "date >
datetime('2018-08-23T00:00:00') AND date < datetime('2018-09-07T00:00:00')"
```

```
plaso - psort version 20210412
Storage file          : plaso.dump
Processing time       : 00:11:33

Events:        Filtered      In time slice   Duplicates     MACB grouped          Total
               2020376                   0          530           454960        2481274
Identifier          PID    Status          Memory         Events         Tags          Reports
Main                219    exporting       171.6 MiB      460898 (0)     0 (0)         0 (0)

Processing completed.
```

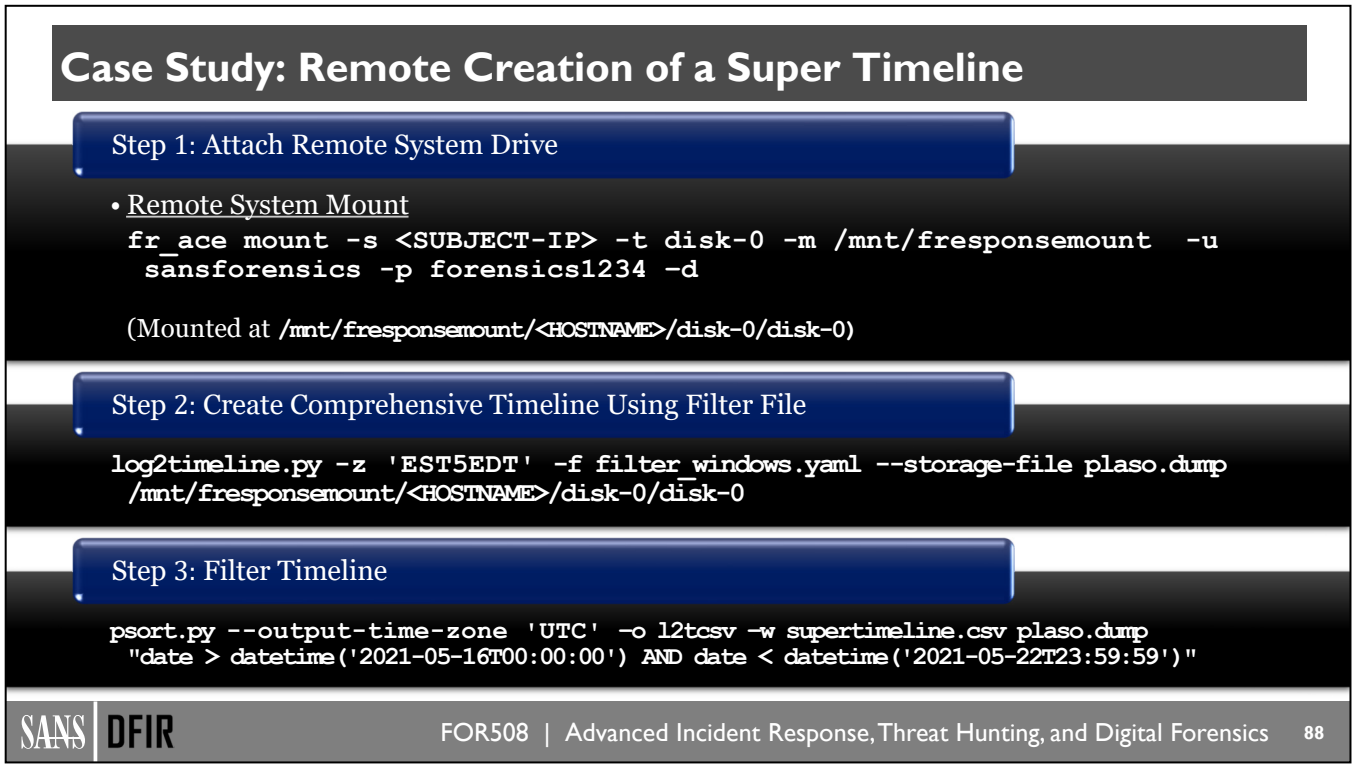


Putting it all together, imagine an incident occurring on a web server which you would like to investigate. With little or no context and the need for speed, you might choose to create a limited timeline to perform alert validation and an initial investigation. You reach across the network and export a collection of triage files from the suspect system.

Step 1: log2timeline is run on the collection of triage files, focusing specifically on Windows event logs and IIS web server logs.

Step 2: We append the contents of the Master File Table (MFT) previously parsed by the external tool MFTEcmd into the plaso.dump database. This step is a little unusual and worth discussing. While log2timeline includes a filesystem parser (named filestat), it is designed to only provide information from files being actively parsed by the Plaso engine (in this example only the event logs and IIS logs). To include the entire contents of the MFT, we must parse it with an external tool like MFTEcmd, which outputs in mactime body file format. We then use the "mactime" plugin within log2timeline to convert and append that data to the plaso.dump datastore. This is a neat trick allowing us to take better advantage of data available in triage images when full disk images are not desired or available. You will see this technique again in an upcoming exercise.

Step 3: psort.py sorts, filters and creates a super timeline is CSV format for review.



In this case study, we see remote timeline creation using the tool F-Response. F-Response provides read-only access to full disks and volumes over the network. Using that to our advantage, we can connect to a remote system, identify the resulting mount point, and use log2timeline in any configuration to create a timeline for review. As collection over the network can be slow, we use a filter file in this example to limit what will be parsed on the remote system.

Step 1: Attach remote system drive using F-Response

```
fr_ace mount -s <SUBJECT-IP> -t disk-0 -m /mnt/fresponsemount -u
sansforensics -p forensics1234 -d
(Mounted at /mnt/fresponsemount/<HOSTNAME>/disk-0/disk-0)
```

Step 2: Create a comprehensive timeline

```
log2timeline.py -z 'EST5EDT' -f filter_windows.yaml --storage-file
plaso.dump /mnt/fresponsemount/<HOSTNAME>/disk-0/disk-0
```

Step 3: Filter and de-duplicate timeline

```
psort.py --output-time-zone 'UTC' -o l2tcsv -w supertimeline.csv
plaso.dump "date > datetime('2021-05-16T00:00:00') AND date <
datetime('2021-05-22T23:59:59')"
```

Timeline Analysis Agenda

- Timeline Analysis Overview
- Filesystem Timeline Creation and Analysis
- Introducing the Super Timeline
- Targeted Super Timeline Creation
- Filtering the Super Timeline
- Super Timeline Analysis

This page intentionally left blank.

Super Timeline Analysis

Understanding and Parsing the Super Timeline Output File

This page intentionally left blank.

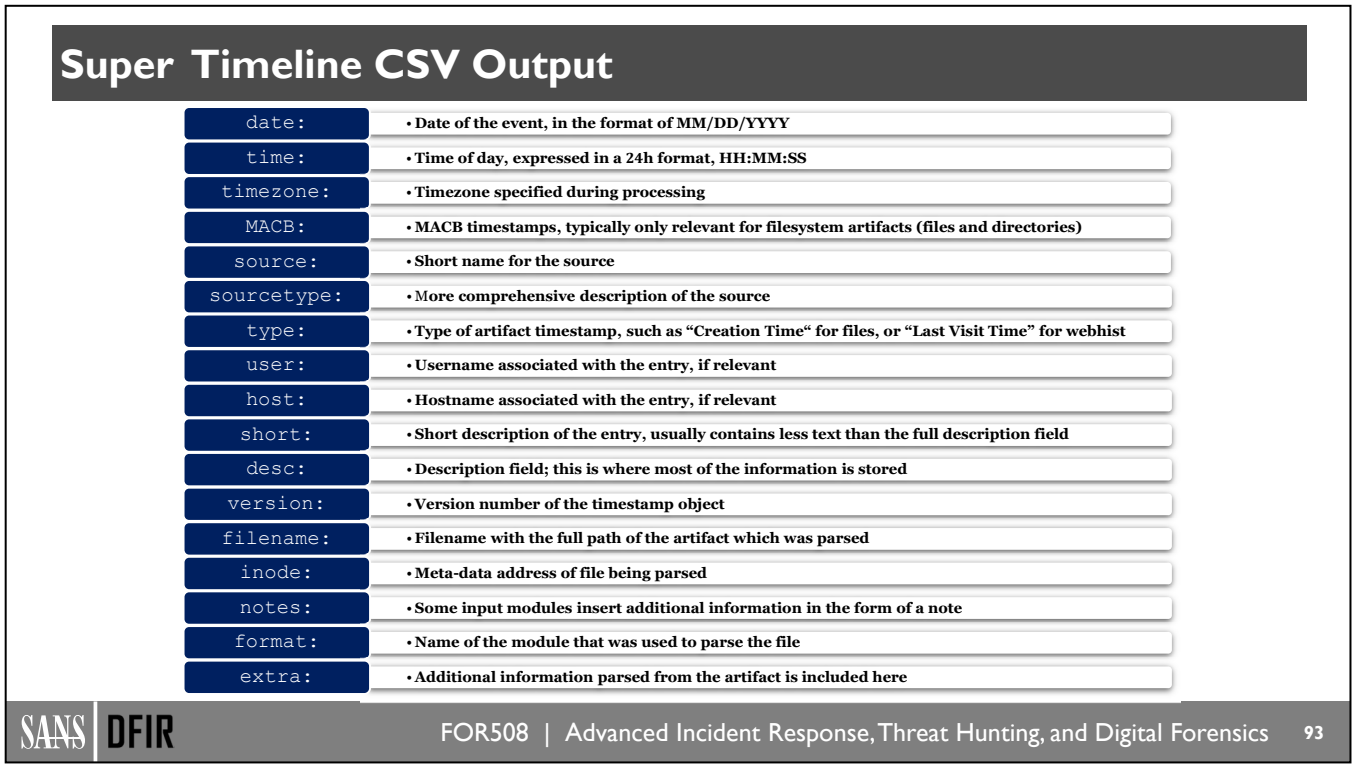
Timeline Output (CSV)

	A	B	F	G	K
1	date	time	sourcetype	type	desc
58462	8/30/2018	22:12:44	Mactime Bodyfile	Content Modification Time; C	c:/Quarantine/7e281e12c2c1fc0.bup
58463	8/30/2018	22:12:44	Mactime Bodyfile	Metadata Modification Time	c:/Windows/SysWOW64/WindowsPowerShell/v1.0/powershell.exe
58464	8/30/2018	22:12:44	Mactime Bodyfile	Metadata Modification Time	c:/Windows/System32/WindowsPowerShell/v1.0/powershell.exe
58465	8/30/2018	22:12:44	WinEVTX	Creation Time	[1002 / 0x03ea] Source Name: Microsoft-Windows-KnownFolders Strings: ['0x80070002' '{B97D20BE
58466	8/30/2018	22:12:44	WinEVTX	Creation Time	[1002 / 0x03ea] Source Name: Microsoft-Windows-KnownFolders Strings: ['0x80070002' '{B4BFCC3A
58467	8/30/2018	22:12:51	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58468	8/30/2018	22:12:51	WinPrefetch	Previous Last Time Executed	Prefetch [CMD.EXE] was executed - run count 11 path hints: \WINDOWS\SYSWOW64\CMD.EXE hasl
58469	8/30/2018	22:13:46	WinEVTX	Creation Time	[145 / 0x0091] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration' 'http://schemas.r
58470	8/30/2018	22:13:46	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration'] Computer Name:
58471	8/30/2018	22:13:50	WinEVTX	Creation Time	[4648 / 0x1228] Source Name: Microsoft-Windows-Security-Auditing Message string: A logon was a
58472	8/30/2018	22:13:51	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58473	8/30/2018	22:13:51	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['EventDelivery'] Computer Name
58474	8/30/2018	22:14:02	Mactime Bodyfile	Content Modification Time; C	c:/Windows/Temp/Perfmon/p.exe
58475	8/30/2018	22:14:02	AppCompatCache	File Last Modification Time	[HKEY_LOCAL_MACHINE\System\ControlSet001\Control\Session Manager\AppCompatCache] Cach
58476	8/30/2018	22:14:02	AppCompatCache	File Last Modification Time	[HKEY_LOCAL_MACHINE\System\ControlSet001\Control\Session Manager\AppCompatCache] Cach
58477	8/30/2018	22:14:51	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58478	8/30/2018	22:15:06	WinEVTX	Creation Time	[145 / 0x0091] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration' 'http://schemas.r
58479	8/30/2018	22:15:06	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration'] Computer Name:
58480	8/30/2018	22:15:18	WinPrefetch	Previous Last Time Executed	Prefetch [CMD.EXE] was executed - run count 11 path hints: \WINDOWS\SYSWOW64\CMD.EXE hasl
58481	8/30/2018	22:15:18	WinPrefetch	Last Time Executed	Prefetch [P.EXE] was executed - run count 1 path hints: \WINDOWS\TEMP\PERFMON\P.EXE hash: 0
58482	8/30/2018	22:15:28	Mactime Bodyfile	Content Modification Time; C	c:/Windows/Prefetch/P.EXE-1209D82B.pf
58483	8/30/2018	22:15:52	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58484	8/30/2018	22:16:26	WinEVTX	Creation Time	[145 / 0x0091] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration' 'http://schemas.r
58485	8/30/2018	22:16:26	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration'] Computer Name:
58486	8/30/2018	22:16:51	WinEVTX	Creation Time	[4648 / 0x1228] Source Name: Microsoft-Windows-Security-Auditing Message string: A logon was a
58487	8/30/2018	22:16:52	WinEVTX	Creation Time	[4624 / 0x1210] Source Name: Microsoft-Windows-Security-Auditing Message string: An account w

This is what a super timeline looks like after it is created and filtered by **`psort`** and loaded into Excel. Wow, what a sight—hundreds of thousands of rows and almost 20 columns. What does it all mean? How can we analyze this much data by hand and maintain sanity? Our biggest challenge with timeline analysis is data reduction. Analysis with the powerful filtering and search capabilities inherent in spreadsheet programs is much easier than simple grep searches to find patterns and strings.

This section and the coming exercise will teach you some tricks and help you maintain your focus during your examination of the data. Pivot points will help significantly narrow your focus and help find starting points. At the beginning, timeline analysis is difficult. But just like files from the operating system, you will not have to examine all 90,000+ of them individually and you will also not have to examine the hundreds of thousands of entries present in a super timeline. Here is what you need to know to survive your first timeline analysis.

	A	B	F	G	K
1	**date**	**time**	**sourcetype**	**type**	**desc**
58462	8/30/2018	22:12:44	Mactime Bodyfile	Content Modification Time; C	c:/Quarantine/7e281e12c2c1fc0.bup
58463	8/30/2018	22:12:44	Mactime Bodyfile	Metadata Modification Time	c:/Windows/SysWOW64/WindowsPowerShell/v1.0/powershell.exe
58464	8/30/2018	22:12:44	Mactime Bodyfile	Metadata Modification Time	c:/Windows/System32/WindowsPowerShell/v1.0/powershell.exe
58465	8/30/2018	22:12:44	WinEVTX	Creation Time	[1002 / 0x03ea] Source Name: Microsoft-Windows-KnownFolders Strings: ['0x80070002' '{B97D20BE
58466	8/30/2018	22:12:44	WinEVTX	Creation Time	[1002 / 0x03ea] Source Name: Microsoft-Windows-KnownFolders Strings: ['0x80070002' '{B4BFCC3A
58467	8/30/2018	22:12:51	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58468	8/30/2018	22:12:51	WinPrefetch	Previous Last Time Executed	Prefetch [CMD.EXE] was executed - run count 11 path hints: \WINDOWS\SYSWOW64\CMD.EXE hash
58469	8/30/2018	22:13:46	WinEVTX	Creation Time	[145 / 0x0091] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration' 'http://schemas.r
58470	8/30/2018	22:13:46	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration'] Computer Name:
58471	8/30/2018	22:13:50	WinEVTX	Creation Time	[4648 / 0x1228] Source Name: Microsoft-Windows-Security-Auditing Message string: A logon was a
58472	8/30/2018	22:13:51	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58473	8/30/2018	22:13:51	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['EventDelivery'] Computer Name
58474	8/30/2018	22:14:02	Mactime Bodyfile	Content Modification Time; C	c:/Windows/Temp/Perfmon/p.exe
58475	8/30/2018	22:14:02	AppCompatCache	File Last Modification Time	[HKEY_LOCAL_MACHINE\System\ControlSet001\Control\Session Manager\AppCompatCache] Cach
58476	8/30/2018	22:14:02	AppCompatCache	File Last Modification Time	[HKEY_LOCAL_MACHINE\System\ControlSet001\Control\Session Manager\AppCompatCache] Cach
58477	8/30/2018	22:14:51	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58478	8/30/2018	22:15:06	WinEVTX	Creation Time	[145 / 0x0091] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration' 'http://schemas.r
58479	8/30/2018	22:15:06	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration'] Computer Name:
58480	8/30/2018	22:15:18	WinPrefetch	Previous Last Time Executed	Prefetch [CMD.EXE] was executed - run count 11 path hints: \WINDOWS\SYSWOW64\CMD.EXE hash
58481	8/30/2018	22:15:18	WinPrefetch	Last Time Executed	Prefetch [P.EXE] was executed - run count 1 path hints: \WINDOWS\TEMP\PERFMON\P.EXE hash: 0
58482	8/30/2018	22:15:28	Mactime Bodyfile	Content Modification Time; C	c:/Windows/Prefetch/P.EXE-1209D82B.pf
58483	8/30/2018	22:15:52	WinEVTX	Creation Time	[5857 / 0x16e1] Source Name: Microsoft-Windows-WMI-Activity Strings: ['WmiPerfInst' '0x0000000
58484	8/30/2018	22:16:26	WinEVTX	Creation Time	[145 / 0x0091] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration' 'http://schemas.r
58485	8/30/2018	22:16:26	WinEVTX	Creation Time	[132 / 0x0084] Source Name: Microsoft-Windows-WinRM Strings: ['Enumeration'] Computer Name:
58486	8/30/2018	22:16:51	WinEVTX	Creation Time	[4648 / 0x1228] Source Name: Microsoft-Windows-Security-Auditing Message string: A logon was a
58487	8/30/2018	22:16:52	WinEVTX	Creation Time	[4624 / 0x1210] Source Name: Microsoft-Windows-Security-Auditing Message string: An account w



The default output of psort is CSV output (using the "dynamic" output module). However, it produces a small subset of fields available from the log2timeline parsing engine. We prefer to use the "l2tcsv" output module for additional fields. Considering the massive amount of information that a super timeline includes, it helps to have more fields available for description information.

The following is a breakdown of the fields available using the "l2tcsv" output format:

`date:`
: Date of the event, in the format of MM/DD/YYYY

`time:`
: Time of day, expressed in a 24h format, HH:MM:SS

`timezone:`
: Time zone specified during processing

`MACB:`
: MACB meaning of the fields, typically only relevant for filesystem artifacts (files and directories)

`source:`
: Short name for the source. All web browser history is, for instance, WEBHIST, registry entries are REG, simple log files are LOG, and so on

`sourcetype:`
: More comprehensive description of the source. For example, "Chrome History" instead of WEBHIST, or "Registry Key - UserAssist" instead of REG

`type:`
: A description of the timestamp itself. Examples include basic file system timestamp descriptions such as “Last Access Time”, “Content Modification Time”, and “Creation Time”. There are many other descriptions as well, which define actions such as process start times (e.g., “Last Time Executed”), website visit times (e.g., “Last Visited Time”), and file download times (e.g., “File Downloaded”).

`user:`
: Username associated with the entry, if relevant

`host:`
: Hostname associated with the entry, if relevant

`short:`
: Short description of the entry, usually contains less text than the full description field (often too little detail to be helpful)

`desc:`
: Description field; this is where most of the information is stored, the actual parsed description of the entry

`version:`
: Version number of the timestamp object

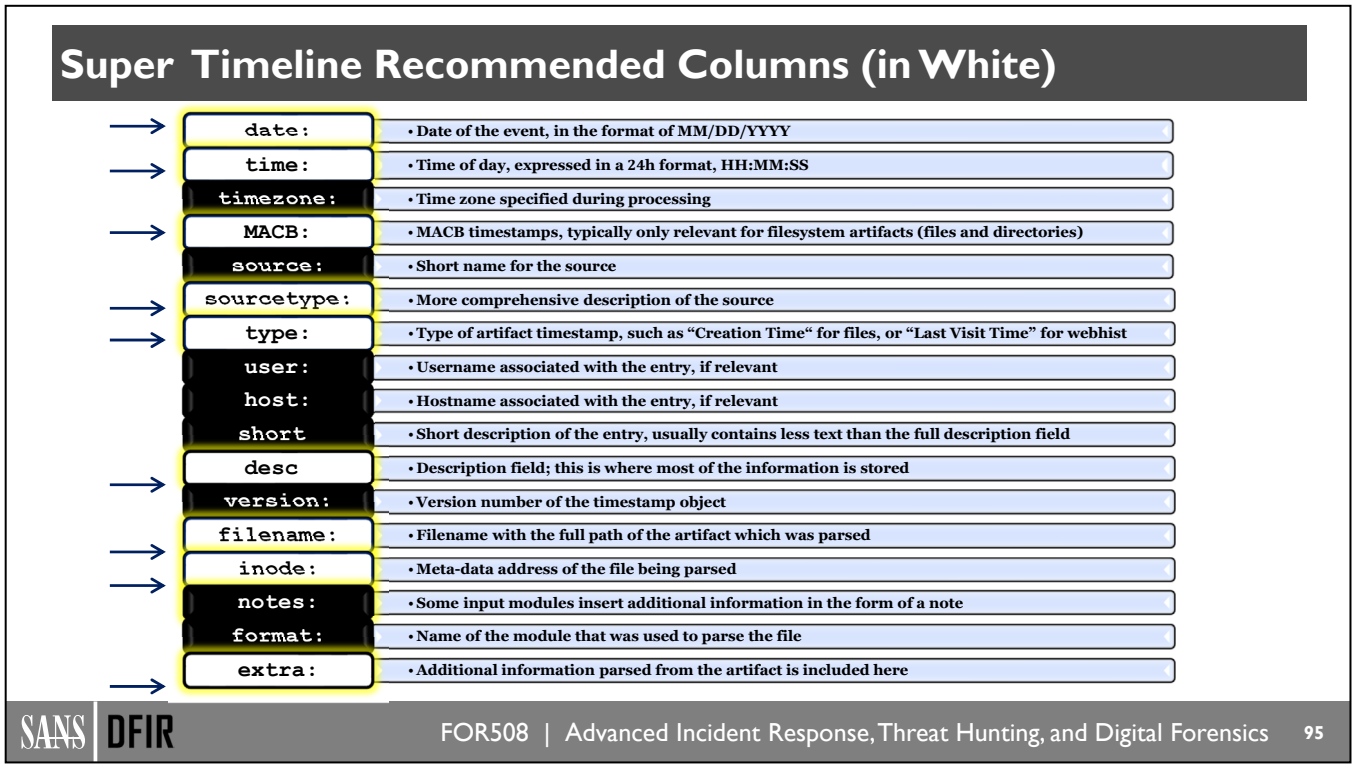
`filename:`
: Filename with the full path of the artifact which was parsed

`inode:`
: Meta-data address of file being parsed

`notes:`
: Some input modules insert additional information in the form of a note

`format:`
: Name of the input module that was used to parse the file

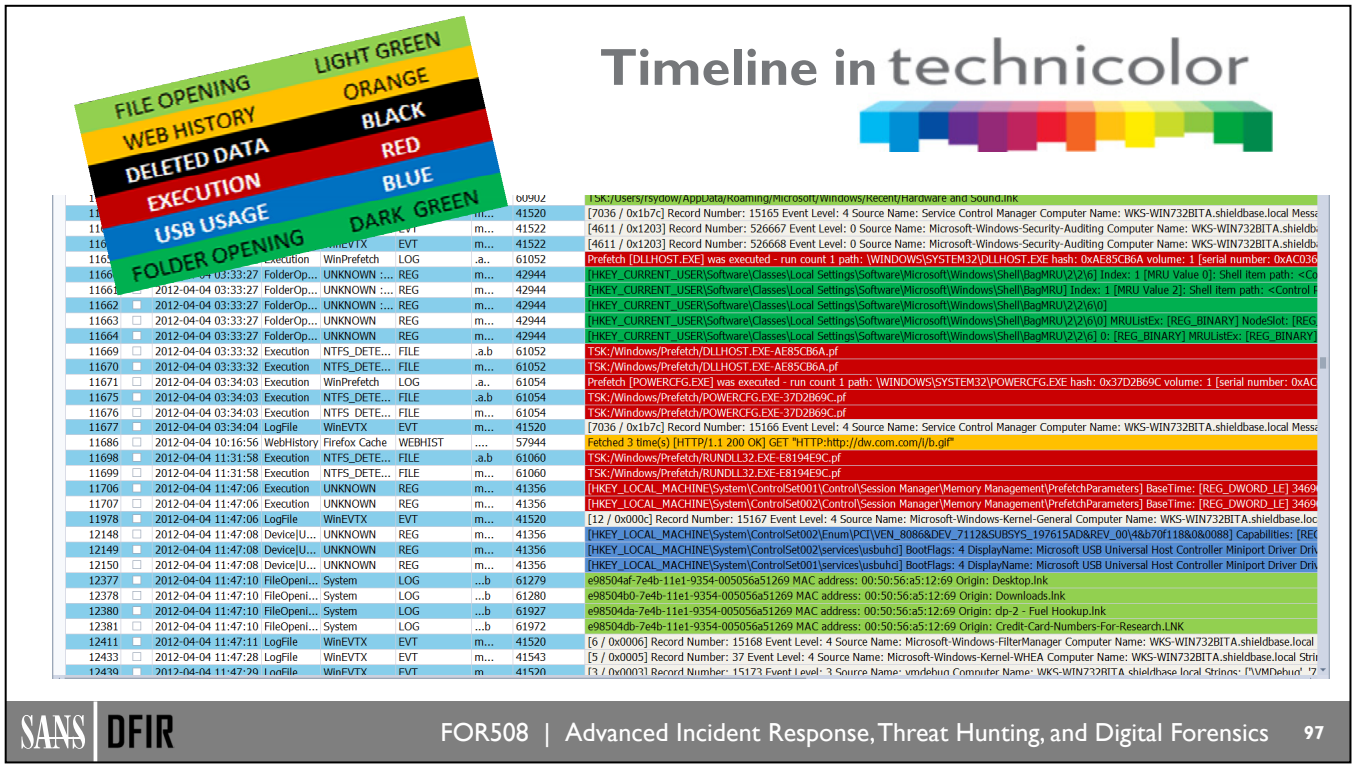
`extra:`
: Additional information parsed from the artifact is included here



The Super Timeline/Plaso output format contains many more columns than the standard "body" forensic timeline format. This can prove challenging when viewed on a system with a small screen. To maximize your screen real estate, we recommend starting with only the columns shown here in white. As you get more experience or start to perform more advanced analysis, other columns may become relevant. As an example, if you are analyzing timelines from multiple systems simultaneously, the "user" and "host" columns become useful. The "Inode" column is useful if you plan to dig deeper into MFT records or other forensic artifacts like the $UsnJrnl but can be hidden during times you need more screen real estate. The "extra" column is not mandatory, but sometimes holds additional information parsed by the plugin.

Feel free to experiment and see which set of columns work best for your cases. Also note that the viewer we will be using in class, Timeline Explorer, includes a "details" view allowing all columns to be viewed in one display, and that display can be moved to a second monitor, so you have all of the information at your fingertips when performing analysis. Double-click on a row to see the "details" view.

Date:	• Date of the event, in the format of MM/DD/YYYY
Time:	• Time of day, expressed in a 24h format, HH:MM:SS
Timezone:	• Time zone specified during processing
MACB:	• MACB timestamps, typically only relevant for filesystem artifacts (files and directories)
source:	• Short name for the source
sourcetype:	• More comprehensive description of the source
type:	• Type of artifact timestamp, such as “Creation Time“ for files, or “Last Visit Time” for webhist
user:	• Username associated with the entry, if relevant
host:	• Hostname associated with the entry, if relevant
short	• Short description of the entry, usually contains less text than the full description field
desc	• Description field; this is where most of the information is stored
version:	• Version number of the timestamp object
filename:	• Filename with the full path of the artifact which was parsed
Inode:	• Meta-data address of the file being parsed
notes:	• Some input modules insert additional information in the form of a note
format:	• Name of the module that was used to parse the file
extra:	• Additional information parsed from the artifact is included here



Performing analysis within a spreadsheet is a decent starting point, but there are more powerful alternatives. SANS instructor Eric Zimmerman created such an alternative. His dedicated tool (aptly named Timeline Explorer), is tailor-made for processing timelines (as well as nearly any CSV or Excel file). [1] Here we see what our timeline looks like once loaded into Timeline Explorer. Multiple timelines can be opened at once (each opens into a separate tab) and data is automatically color-coded according to the type of artifact. A legend is available under the "Help" menu showing what each color indicates. In the example on this slide, you can tell when USB devices were utilized in the timeline (blue). You can also tell when files were opened (green) and when programs were executed (red).

Timeline Explorer can open nearly any CSV or Excel document (first worksheet only) and display the contents in a grid. This data is read-only and cannot be changed by someone viewing the data. It supports a wide variety of formats including Autoruns, Mactime-generated timelines, Plaso super timelines, and all of Eric Zimmerman's command line tools (CSV output).
Shortcuts
CTRL-T: Tag or untag selected rows
CTRL-D: Bring up Details (for use with super timelines)
CTRL-C: Copy selected cells (and headers) to clipboard (the behavior can be adjusted via the Tools menu)
CTRL-F: Show Find dialog
CTRL-Down: Select last row
CTRL-SHIFT-A: Select all values in current column

Wildcards
Wildcards are supported in column filters when using the LIKE operator. % matches anything and _ matches a single character. CONTAINS implies wildcards and is the default, but using LIKE along with wildcard operators can often be more precise.

Tagging allows for selecting a subset of data, filtering on it, then exporting to Excel format using the File menu. This technique allows you to get data out of the tools for reporting purposes. Tagging can also be used to notate interesting results and go back to them as needed.

Double-clicking any item in a Plaso super timeline brings up the Details window. It allows for more comprehensive inspection of data in a non-horizontal format. This format is particularly useful when reviewing event log entries. There are options to make the Details window the topmost window and buttons to move to the next or previous record.

Search options menu
The location of the search options menu varies depending on the version of Timeline Explorer. This menu controls how the global search bar operates. As an example, you can set search to default to "Filter" or "Search". Filtering can be very powerful as any rows not containing the search term used are filtered out of the view. If Search is selected instead, search hits will be highlighted, but no rows will be filtered out.

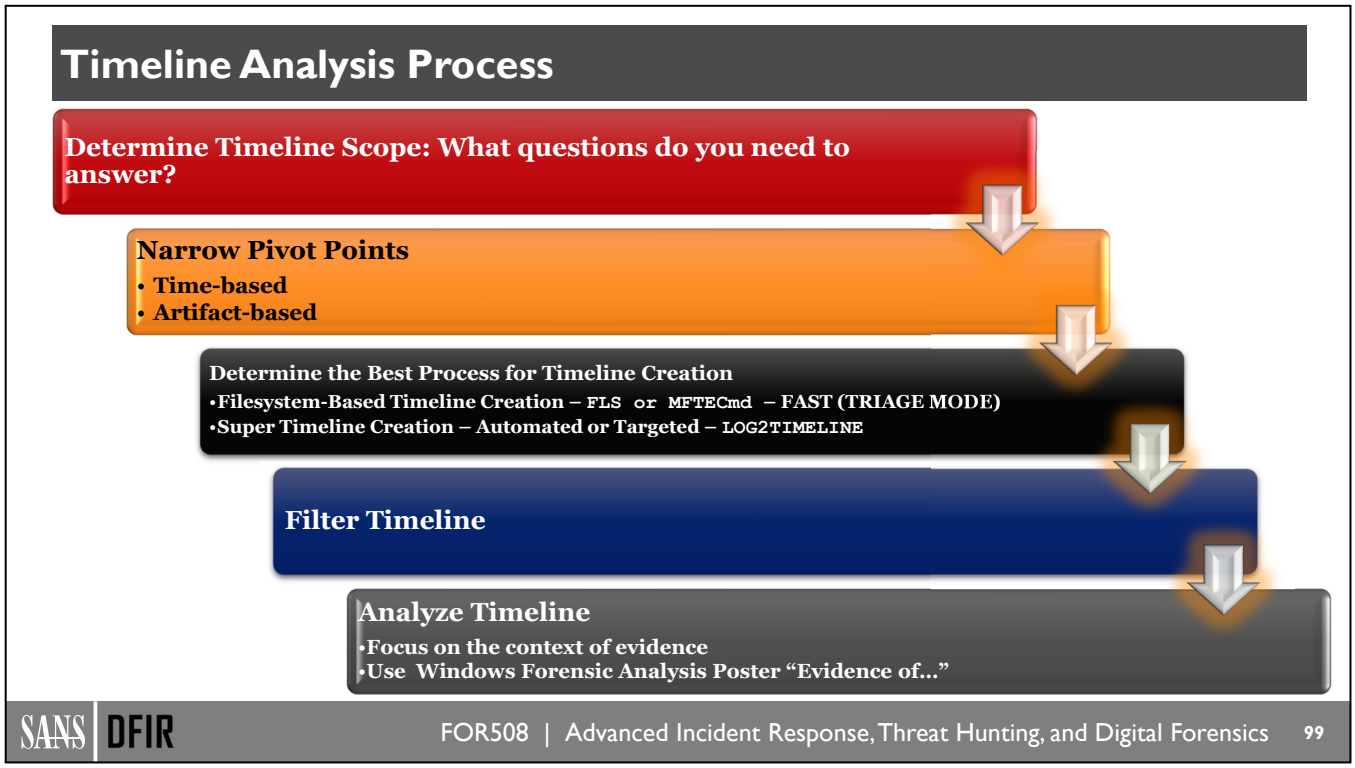
Layouts
For all supported file types, layout files are saved and loaded as needed. The layout files contain settings for that file type, such as which columns are shown, the order of columns, conditional formatting rules, and so on. These files are stored in the "Layout" folder where Timeline Explorer is installed on the system. Layout files are what tells the tool how to colorize specific lines.

Your first several tries at accomplishing timeline analysis will be challenging due to learning how to filter thousands of lines of data while learning and memorizing exactly what artifact patterns are useful. Typically, I have found one thing to be true: Unless I need them, I generally remove any temporary internet files—there are just too many. You will start to have your own preferences as well. Use the colors to draw your attention to interesting parts of the timeline, but don't focus solely on them! Many interesting artifacts (like event logs) are not colored because doing so would be visually overwhelming.

Eric Zimmerman is very open to feedback and feature requests from students. If there is something you would like to see in Timeline Explorer, please reach out to Eric!

For those curious as to the origins of this analysis technique, the original timeline colorization was released as an Excel template. If you prefer using Excel, you can find the process and template on the SANS website.[2]

[1] Eric Zimmerman's Tools: https://for508.com/8t6pb
[2] Digital Forensic SIFTing: Colorized Super Timeline Template: http://for508.com/relfp



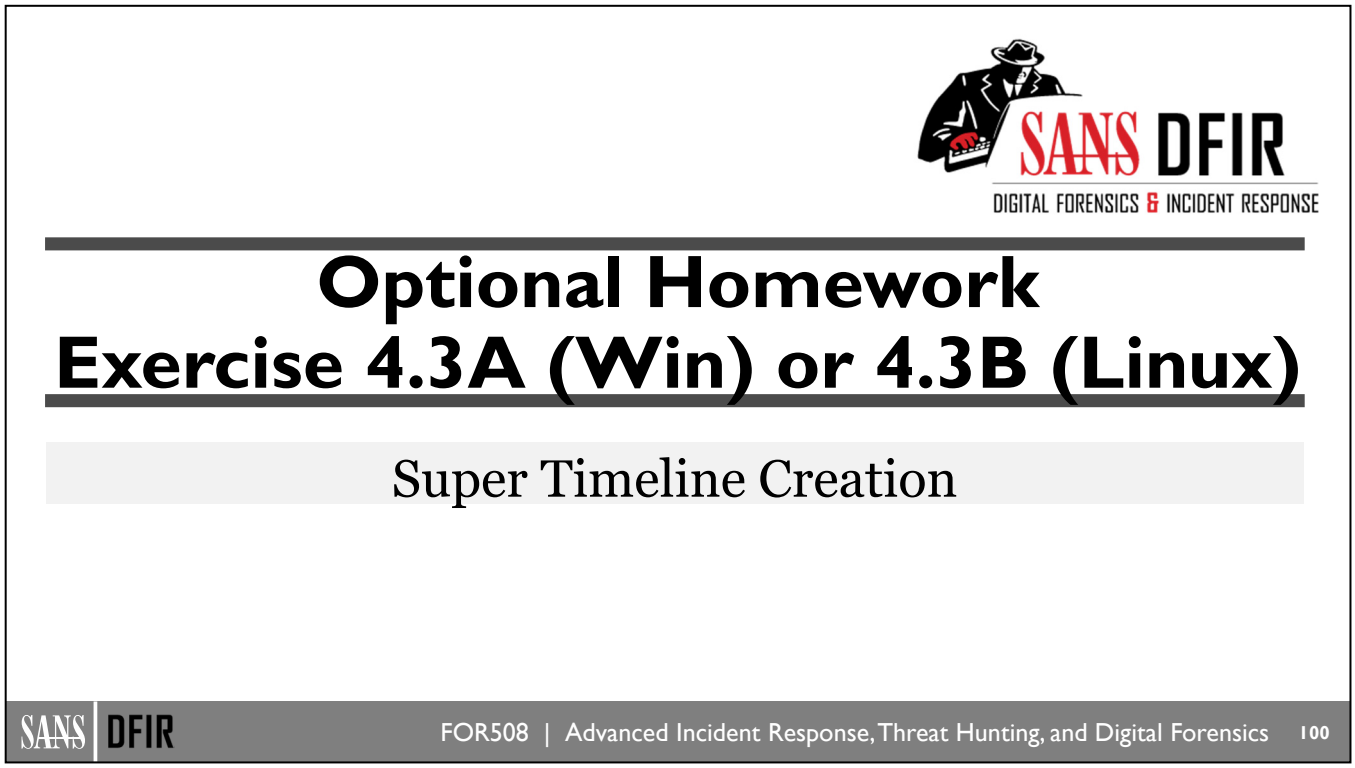
Determine Timeline Scope through analysis of the key questions and the case type. Generally, you can identify that the activity occurred between date A and date B. Narrowing the scope will be important in managing your overall data.

Narrow Pivot Points through determining the closest time around when you think the incident occurred (time-based) ,or identifying a key file, user account, or other artifact that might have been used in the activity in question (artifact-based).

Determine the Best Process for Timeline Creation by looking at what data sources you need. A super timeline is preferred if you do not know what you are looking for and you might need to include everything. However, if you can predict the data necessary to solve your case, a targeted super timeline or filesystem timeline might suffice.

Filter Timeline using your scope, de-duplicate, and eliminate data you do not need to examine. Consider using keywords to identify relevant data (pivot points) for your analysis.

Analyze Timeline through **focusing on the context of evidence** discovered. Look before and after each pivot to determine user activity. **Use the Windows Forensic Analysis Poster "Evidence of" items** to help with analysis.



This page intentionally left blank.

Exercise 4.4

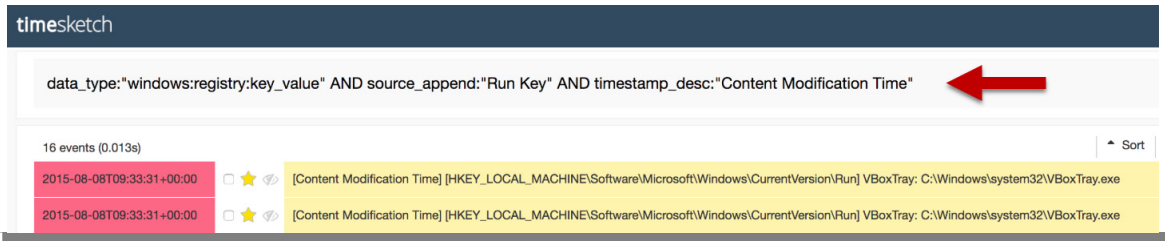
Super Timeline Analysis

Average Time: 45 Minutes

This page intentionally left blank.

Scaling Timeline Analysis

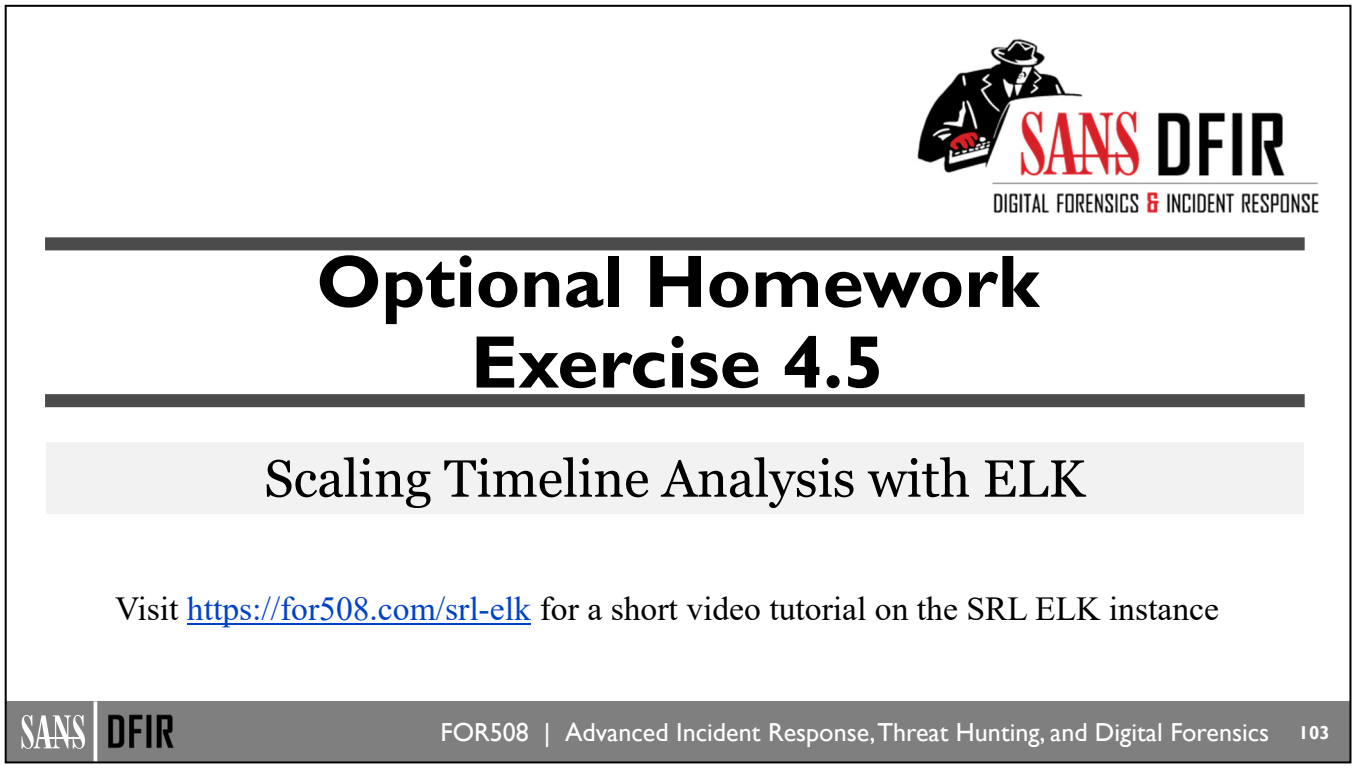
- Search timelines for IOCs
  - yara_match.py
- Use a database to investigate multiple timelines
  - Splunk
  - ELK
- Timesketch



Hopefully, by this point, you see the value of timeline analysis and its ability to provide quick answers to forensic questions. Now imagine the ability to get those questions answered at scale. Here we see two ways to attack the challenge of performing timeline analysis across many timelines simultaneously. One easy solution is to look for obvious indicators of compromise across a collection of timeline files. This could be as simple as performing regular expression searches across a folder of timelines or utilizing an IOC framework like YARA. Even basic checks like looking for executables in the $Recycle.Bin, svchost.exe in a folder other than Windows\System32, known recon tool execution, or the names of known bad files and registry keys could quickly identify timelines worth looking deeper into. Kristinn Gudjonsson, the creator of Plaso, wrote a script named **`yara_match.py`** to facilitate searching a super timeline for a list of YARA signature matches.[1]

Loading multiple timeline files into a database is a more robust approach. Splunk (free or commercial version) makes a great choice for indexing and searching timelines.[2] Dave Herrald released a Splunk App to make it easy to ingest timelines in the log2timeline format.[3] ELK, an acronym for elasticsearch + logstash + kibana, is a very popular (and free) software stack used to ingest massive amounts of data, index and create dashboards to facilitate analysis and data visualization. Plaso ships with an output format designed to be used with ELK (the output option is named “elastic”).[4] Getting multiple timelines in a database allows searching for patterns across many timelines at once. Taking things a step further, Johan Berggren created a front-end to log2timeline and elasticsearch named timesketch.[5] Timesketch allows multiple investigators to collaborate across many timelines in real-time all while tagging, annotating and enriching the data. When you are ready to take things to the next level, these projects will help you get there.

[1] l2t-tools/yara_match: https://for508.com/5v687
[2] Using Splunk for Computer Forensics: https://for508.com/nuvhm
[3] Dave Herrald SA_plaso-app-for-splunk: https://for508.com/4flp-
[4] Plaso and elasticsearch: https://for508.com/46auk
[5] Google timesketch, collaborative forensic timeline analysis: https://for508.com/uft34



Visit https://for508.com/srl-elk for a short video tutorial on the SRL ELK instance.

This page intentionally left blank.

COURSE RESOURCES AND CONTACT INFORMATION

Here is my lens. You know my methods. -Sherlock Holmes

AUTHOR CONTACT
rlee@sans.org
http://twitter.com/robtlee

ctilbury@sans.org
http://twitter.com/chadtilbury

mpilkington@sans.org
https://twitter.com/mikepilkington

SANS INSTITUTE
11200 Rockville Pike, Suite 200
N. Bethesda, MD 20852
301.654.SANS(7267)

DFIR RESOURCES
digital-forensics.sans.org
Twitter: @sansforensics

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.